DISSERTATION

A TALE OF 'T' METRICS:
CHOOSING TRADEOFFS IN MULTIOBJECTIVE PLANNING

Submitted by

Mark Roberts

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Fall 2013

Doctoral Committee:

Advisor: Adele Howe
Co-Advisor: Indrajit Ray

Darrell Whitley
Daniel Turk

ABSTRACT

A TALE OF 'T' METRICS:
CHOOSING TRADEOFFS IN MULTIOBJECTIVE PLANNING

Search-based planning systems are often evaluated according to their computational efficiency and final plan quality under the assumption of producing a single plan that is evaluated according to a single metric. In this context, a planner is preferred if it produces a better quality solution with the same (or fewer) computational resources. This evaluative focus has naturally led the planning community to design planning systems based on the tradeoff between efficiency and quality, and has led to techniques that improve performance along one or both of the quality-efficiency dimensions. The end result is continuing advancement of the state-of-the-art for applications needing the best quality (sometimes optimal) plan. In general, planners get faster at finding that single best solution.

We present a motivating application – personalized cybersecurity – that challenges the single-solution, single-metric assumption. First, this application requires the planner to generate plan sets instead of a single solution. Generating plan sets adds a new evaluation dimension, frequently called diversity, to the already existing dimensions of efficiency and quality. Further, our motivating application requires that plans are evaluated according to multiple quality metrics that interact in subtle ways. The two challenges of producing plan sets and evaluating under multiple metrics form the two research directions of this work. We seek to understand the implications of these two directions in how we design and evaluate planners.

A planner's performance is typically evaluated along several dimensions: its *coverage* over the number of problems it solved, the *quality* of its final plan for a given problem, and its computational *efficiency* of producing each plan. Computational efficiency is usually measured using time or memory consumption. Plan quality is usually a one-dimensional criterion that uses either the length of the plan or a sum over the cost of its actions. The more

recent evaluation focus of *diversity* is usually measured as the distance between plans using the set difference or set intersection of their actions. Recent work has extended planning systems to produce diverse plan sets [Ngu+12; CMA11; Sri+07; Rob+12]. But relatively little research has been done to understand the tradeoffs of plan set diversity in the context of multiple quality metrics that may also interact with each other. Matters get worse when diversity and quality interact.

Our approach is to tackle each research direction independently before combining them. Thus, we first examine producing single-metric plan sets before examining metric interaction while producing single plans. Finally, we combine the two directions to examine producing plan sets when the same metrics interact. We evaluate the tradeoffs along three evaluation axes of quality, efficiency, and diversity using many of the evaluation metrics that are in the literature. The existing diversity metrics have two shortcomings. Plan distance does not provide comparison *between* two plan sets produced by different approaches, so we create a metric, *overlap*, that takes the set intersection of two plan sets. Plan distance also does not characterize the distinctness of plans *within* a plan set. We create two diversity metrics designed to alleviate this. We found in our studies that plans within a plan set could be subsets of each other. The first metric, *uniqueness*, captures the way in which plans do not subsume each other by removing any padded or permuted plans. The second metric, the *parsimony ratio*, characterizes how verbose a planner is for a given plan π_l with respect to a minimal plan, π_k. We obtain π_k (for as many problems as possible) by running A* while restricting the allowed operators to only those in π_l.

In our study of producing single-metric plan sets, we focus on variants of A*, a common base algorithm for many state-of-the-art planners. A* is designed to produce a single solution using an estimate of the cost to achieve a solution. Our study of single-metric diversity includes four algorithms based on A* and one full planning system. To maintain a fair comparison, we implement the algorithms in the same planning framework. The algorithms include random walk search [XNM12], modifying the heuristic to drive diverse search [CMA11], a new algorithm, Iterated Tabu A* (*ITA*), that encourages plan set diversity using

a state-based Tabu list [Rob+12], and a hybrid of the heuristic and tabu approaches. We complement our analysis of the algorithms with a full planning system, called LPG-diffmax [Ngu+12], that is designed to produce diverse plan sets. We run these five approaches on several benchmark domains from the International Planning Competitions (IPCs).

We find that approaches that generate the highest diversity with the highest uniqueness also produce plans that sacrifice plan quality and reduce search efficiency. Our findings challenge the approach of using diversity as a metric to both drive search and evaluate planner performance. These approaches produce plans that appear unique but drive high diversity by repeating action sequences that do not lead to the goal; the plans are not parsimonious. Although the full planning system solves more problems, it does not always produce more diverse plans (nor more unique plans) than the simpler algorithmic approaches. Finally, our findings suggest that selecting the best approach for generating alternatives depends on the evaluating metric, which ultimately depends on the needs of the application for which those alternatives are generated.

To assess the impact of metric interactions on search behavior, we start with single-solution search as a baseline. Since there are very few domains with metric interaction, we create a synthetic domain that allows us to vary the interaction of two metrics, x and y, in a weighted objective function, z, while controlling for plan-length. We use a common variant of A*, called A^*_ϵ, that provides bounds on the cost of the solution returned. One of the more surprising findings is that A^*_ϵ search performs quite poorly when minimizing collinear functions ($y = x$ and $y = sigmoid(x)$), which suggests that researchers should avoid combining x and y when they are (nearly) collinear. A^*_ϵ works well for curvilinear functions such as polynomials and an "inverted" sigmoid. However, scaling the metrics appears to dramatically reduce search effectiveness. Poorer performance occurs when the metrics were uniformly scaled to control, at least in part, for plan-length correlation. Search performance also degraded as one metric was weighted more heavily. These findings have implications not only for the general use of weighted objective functions in A* search (and its variants), but more specifically for using any weighted combination of diversity and quality.

Bolstered by our understanding of how metric interaction impacts single-objective, single-solution search, we create a new algorithm, Multi-Queue A* (*MQA*), that is designed to produce diverse alternatives under multiple objectives. *MQA* manages each quality metric in its own queue, thus alleviating (some of) the issues of scale and interaction caused by combining metrics in a single objective function. We drive diversity by using what we call a "parsimony queue" that first minimizes the heuristic estimate and then maximizes diversity. We find that *MQA* with the parsimony queue indeed results in more unique solutions than other algorithmic approaches while maintaining good parsimony. We also find that it can produce a better spread of solutions along the solution front of the synthetic problem. However, in cases where quality and diversity are collinear, it achieves better diversity while sacrificing solution quality. Finally, we find that *ITA* still produces the most unique solutions for the security domain of any approach we examined.

Our findings synthesize the recent literature on generating diverse alternatives. In practical terms, they suggest that the objectives of the application should direct the choice of the algorithm and choice of the search control mechanisms for generating plan sets on a pareto-optimal front. Contrary to the conventional wisdom of modifying the heuristic to improve search, we find heuristic modifications alone do not account for the success of producing diverse plan sets. Instead, we show that parsimony must be a central concern and that using a portfolio-like multi-queue approach can lead to good results – depending on the application. We show that targeted changes to the A* algorithm can improve search for specific applications, but that such changes must be tailored to the application for which they are intended.

ACKNOWLEDGMENTS

Getting this far in any pursuit requires the support of many people. Foremost, I want to thank my primary advisor, Dr. Adele Howe, for all her mentoring and support. Her professional and personal guidance over the many years of my journey has shaped me for the better. Words cannot express the depth of gratitude I feel for her. I also want to thank Dr. Indrajit Ray for his guidance, especially in his encouragement to reconsider an academic path for my career.

Along my trek, I was fortunate to consider as my mentors and colleagues many bright and engaging people at CSU, SRI, NASA, and HP. Each one of them gave me insight into how to think more clearly, how to be a better professional, and how to be a better human. I want to thank the many people in the ICAPS community for many productive and fun conversations on all topics research and personal. Many of these individuals provided feedback on my work, which only helped to improve its quality.

I have been privileged to receive support for this project from the National Science Foundation, and this material was supported by the National Science Foundation under Grant No. 0905232. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Finally, I sincerely appreciate the support of my family and friends. I especially thank my friends from Fort Collins, who enjoyed the outdoors or a beer with me working out the finer issues of research or life. To the many people who have supported me in one way or another, thank you. Your investment as a friend, colleague, mentor, or all three has been an invaluable gift to me. I'll be sure to reinvest it wisely!

TABLE OF CONTENTS

Chapter 1
The 'T' Metrics of Planner Evaluation

Classical Planning is a sub-field of Artificial Intelligence concerned with producing a set of actions that achieve a goal state from an initial state. Figure 1.1 provides an example of a logistics application, the kind of application to which planning is naturally suited. The left side of this figure shows a white truck in City 4 and a package in City 1. The goal state shown on the right indicates that the package is needed in City 3, and, while the truck is also at that location, there is no requirement that the truck end up there (though the truck will always be at that location in this example since the problem will be solved as soon as the package is unloaded). The operators and states form an implicit state transition system that we will formalize in Section 3.1.

The dominant paradigm for producing a plan in Classical Planning is best first search over this implicit state transition system. Often the algorithm is some variant of A*. Best first search proceeds by taking what it believes is the current best state, expanding all possible operators on that best state to produce the successors, and then queuing the successors into a data structure that stores the states that remain to be explored. To ensure that search does not repeat effort on states it has already visited, a closed list keeps track of the states

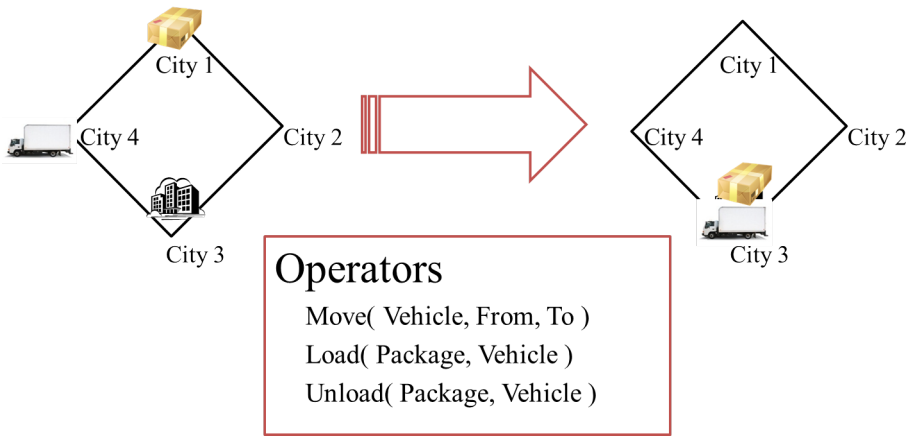

Figure 1.1: An example planning problem where the goal (right) is to move the package from its initial position at City 1 (left) to a location at City 3 using the transition operators `Move`, `Load`, and `Unload`.

that have already been expanded. Best first search is often informed by a heuristic to help it select the more promising states (i.e., those states on the path to the goal). In general, when we talk about search-based planning we mean state-based best first search guided by a heuristic.

Search-based planning has at its core an assumption of producing a single plan according to a single quality metric. This has huge implications for how planners are designed. Often, planners are evaluated on their ability to cover more problems, the efficiency (i.e., time and memory) with which they solve problems, and the quality (i.e., plan length or plan cost) of the final plans. Generally speaking, a more efficient planner will solve more problems and have higher coverage. So, in this evaluative context, one planner is preferred over another planner for a given problem if it produces a better quality solution with the same (or fewer) computational resources. This evaluative focus has naturally led the community to design systems that balance the tradeoff between efficiency and quality, and has led to techniques that improve performance along one or both of the quality-efficiency dimensions. The end result is continuing advancement of the state-of-the-art for applications needing the best quality (sometimes optimal) solution. In general, planners get faster at finding that single best solution. But not every application requires these assumptions.

Our motivating application – personalized cybersecurity – challenges the single-solution, single-metric assumptions upon which many planners are evaluated. In particular, the secu-

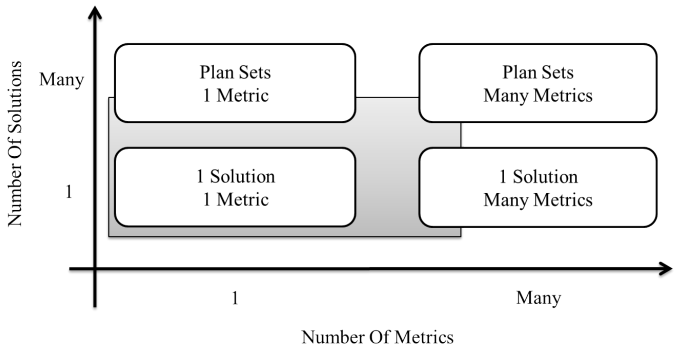

Figure 1.2: The gap of existing planning research: much of the planning literature (the gray inset box) focuses on producing single solutions under a single quality metric.

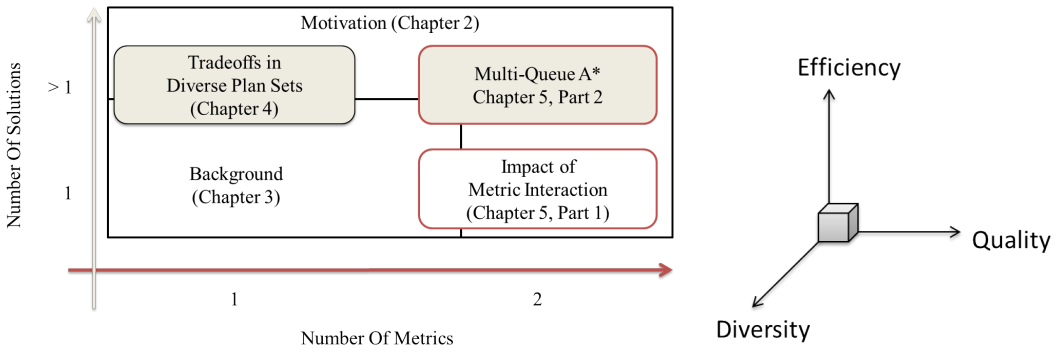

Figure 1.3: The two research directions of this dissertation (left) and the axes of evaluation (right).

rity application requires a planner that generates sets of plans that are evaluated according to a variety of metrics. These two requirements yield the research directions of this dissertation, as shown in Figure 1.2.

The two research directions of multiple solutions and multiple metrics have implications for how to design and evaluate planners. We focus on three axes of evaluation as presented in the right subplot of Figure 1.3. It will still be the case that we care about quality and efficiency, although we will need to adjust several aspects of evaluation. Along the efficiency axis remain concerns about the computational resources (time and memory) that are consumed by the planner. Along the quality axis remain concerns about the "goodness" of the final plan that is produced, where quality is usually measured by a single metric such as the length of the plan or the sum of the action costs. However, the quality axis now includes additional concerns over managing quality in the face of multiple metrics.

The requirement of producing plan sets adds a third evaluation dimension that has been called *diversity* in the planning literature. It is not of great use to have a set of plans that differ only trivially. So along the diversity evaluation axis we are concerned with metrics that assess the distinctness of a solution in a plan set, which has usually been measured in the literature as the set difference between the actions of the plans. Note that this evaluation dimension can be closely tied with how plan sets are produced as well. In reality, considering plan sets raises two questions that are not necessarily related to one another: 1) how to find

solutions and 2) how to evaluate those solutions. However, many approaches in the literature conflate these two questions – the same technique is used to both produce diverse plan sets *and* evaluate those plan sets. Along the diversity axis, we examine plan sets using two distance-based diversity metrics from Nguyen et al. [Ngu+12; Sri+07] as well as Coman and Muñoz-Avila [CMA11]. We show how these evaluation metrics are too similar to the way in which search is driven and can be misleading indicators for good diversity. To complement these distance-based metrics, we add two set-based metrics we call "overlap" and "uniqueness" that both help characterize the contribution of individual plans within a plan set. We also found that many plans can be subsumed by other plans, so we create a new metric called the "parsimony ratio" that measures the extent to which a plan is longer than it needs to be.

Several approaches have shown success in isolated experiments for producing diverse plan sets (e.g., [Ngu+12; CMA11; Sri+07; Rob+12]). It is less clear how these approaches compare to one another, how diversity, quality, and efficiency interact, and how such interactions may have implications for search performance (and thus, of how we evaluate that performance). In this study, we investigate the tradeoffs of producing plan sets along the three axes of diversity, quality, and efficiency. Our analysis leverages existing metrics for evaluating quality and efficiency by using the common measurements already mentioned: search cost, plan length, and plan quality. We are also interested in the interplay of the diversity-quality and diversity-efficiency axes, so we characterize the plan sets using quality and efficiency metrics.

Interactions can take place within the same axis. For example, a manufacturing application may require plans that create products using distinct methods, where each method produces varying levels of pollution while consuming varying levels of resources. Such a tradeoff occurs within the quality axis. Much of the planning literature focuses on a single metric, but less is understood on the interaction of quality metrics.

Interactions can also take place between axes. For example, it might be computationally efficient to compute a low quality plan but take increasingly more computational resources to produce better and better plans. Much of the planning literature has focused on efficiency-

quality tradeoffs, which has led to some very efficient planners that produce a single solution under a single metric evaluation. But less is understood about the tradeoffs of these efficiency and quality metrics with respect to diversity, when more than one plan is being evaluated.

This dissertation examines the performance tradeoffs of moving along one or both of two research directions: producing multiple solutions or assessing plans with multiple metrics. These two research directions are the independent variables of the study, and we tackle each direction in isolation before combining them together. The 'T' metrics of this dissertation fall along three axes; these are the dependent variables of the study. **Efficiency metrics** quantify the computational effort a planner expends to produce a solution. Since much of the computation effort of planners lies in searching for a solution, these metrics include ways to quantify CPU time or memory usage. **Quality metrics** assess the value of a single plan. These metrics tend to be application specific and include ways to quantify the effort of executing the plan. One common plan quality metric, a plan's length, has played a large role in the development of planners. **Diversity metrics** assess the value of a plan set both within a plan set and between plan sets. Examining alternatives is best done with the widest possible set of alternatives, and these metrics try to measure the distance any two plans will be from each other. Together, these metrics interact to create tradeoffs that we characterize. We endeavor to advance the algorithms of planning systems with an analysis-driven approach that leverages the knowledge we gain from our study into these tradeoffs.

Figure 1.3 shows both the two research directions (the independent variables) and the axes of evaluation (the dependent variables) and how the main chapters of this document relate. Chapter 2 more clearly motivates how the security application (and other applications) require these two new directions. Chapter 3 relates our work with the existing literature. Along the vertical axis (in gray), we assess tradeoffs associated with having a planner produce plan sets instead of just a single solution; this is studied in Chapter 4. Along the horizontal axis (in red), we study metric interaction. In order to study metric interaction, we justify and present a synthetic domain in the first half of Chapter 5 and follow with results of running two variants of A* on this domain. Finally, in Chapter 5 we combine the knowledge

we gather into the design of an algorithm called Multi-Queue A* that takes a step toward producing diverse plan sets while managing metric interaction.

The main contributions of this dissertation are as follows. We create new diversity metrics for comparing plan sets and different approaches that generate plan sets. These metrics allow us to analyze the tradeoffs of producing plan sets as well as to analyze search behavior when metrics interact. We show that combining diversity in a weighted metric has the unintended consequence that it often seriously misleads search with the undesirable tradeoffs of decreased efficiency and poor quality. We also show that the weighted linear combination of multiple metrics confounds searching for good quality plans that *balance* the objectives at hand precisely when the metrics interact in unexpected ways or when the metrics are pathologically intertwined (i.e., they fall in the same range but are contradictive). For planning, diversity is at odds with quality because long plans usually imply poor quality, thus further confounding the weighted objective function. This leads us to the perspectives that diverse plans are only worthwhile if they are also parsimonious and that alternative ways of driving diversity are needed. We present two algorithms that attempt to alleviate some of the problems of a weighted objective function. The first, Iterated Tabu A* (*ITA*), extends A* with a Tabu list to encourage exploration of new states. The second, Multi-Queue A* (*MQA*), extends A* to manage distinct queues for each quality metric and diversity metric to maintain parsimony and quality and while encouraging diversity.

Our study synthesizes the recent literature on generating diverse plan sets. As far as we are aware, it is the first study to examine the tradeoffs of contrasting approaches and then propose ways to address those tradeoffs. Since each application will have differing concerns with respect to quality and diversity, our results show that the objectives of the application should direct the choice of the algorithm and choice of the search control mechanisms for generating the plan sets. We show that understanding the interplay of diversity, quality, and efficiency leads to an improved algorithm design for applications such as the security domain and that targeted changes to the A* algorithm can improve search for such applications.

Chapter 2
Motivating Planning Advancements Through Applications

Our work advances planning systems to meet the demands of applications. In this context, a planning system produces a plan, which is sequence of steps (or actions) that achieve some goal; we will present a formal model for planning in Section 3.1. Figure 1.1 provides an example of one kind of plan for a logistics domain where the goal is to move a package from one location to another. In such domains, the planner constructs a sequence that achieves some end that (assumedly) will be executed by some agent in a physical world. A plan can also be a way for a human or software agent to reason about the potential ways an event *might* happen. For example, a plan could represent a sequence of steps leading to a compromised computer system, and the plan itself is never intended to be executed.

The focus problem for our research is a security agent for home computer users. We discuss the security agent and its underlying security model, which presents two distinct challenges for planning: generating alternatives and evaluating plan quality with multiple metrics. Many other applications also require the planner to support exploring alternatives with complex evaluation of the tradeoffs. So we end the chapter with a brief overview of other potential applications that further motivate our research.

2.1 Focus Problem: Cybersecurity For Home Computer Users

The Internet has brought many benefits to users who embrace it. Tools such as email, texting, web pages, search engines, shopping, gaming, social networking, etc., make communication much more efficient and allow for disparate groups of individuals to form regardless of geographical boundaries or social strictures. However, never before has a technology brought so much risk to an individual's security and privacy. A user can engage in apparently innocuous behavior only to discover that private information has been compromised. Even the most knowledgeable computer user may find it challenging to maintain a safe computing environment. Casual computer users may find the array of security choices daunting

in the midst of having to learn new terminology. Sophisticated users demand more fluid interfaces with complex systems and get annoyed by constant, pedantic interruptions, while novice users may need more individualized instruction to deal effectively with potential security threats. Finding a balance between these user groups is challenging, and security software does not cater to the needs of individual users within these groups.

Consider a security agent that is tasked with helping a home computer user mitigate security and privacy breaches. Such an agent would need to perform all of the following tasks: monitor the user/system for new behavior/state, adjust its behavior based on the user and the current user's task, incorporate new exploits from a common security database, adapt to newly installed software, block the most critical vulnerabilities first, offer suggestions of actions to the user to support achieving his/her goals while not breaching security/privacy, and intervene independently to the extent that the user's trust allows.

The agent must be able to offer alternatives if the user is not satisfied with a plan it has selected. For example, the user may be starting to install some peer-to-peer (P2P) software that includes a virus. Suppose the preferred plan is to check that no virus has yet been installed and to halt the installation; however, because the user desires the P2P functionality, an alternative plan of searching for different P2P software and installing a more secure application may be better.

Planning is amenable to the security domain because a security attack proceeds along a set of state transitions (i.e., the system state) that are advanced by actions (i.e., the attacker, the user, or the attacking software). Previous research has shown how a planner can help analysts identify actions that lead to security breaches. Boddy et al. [Bod+] built a mixed initiative planning system that could identify potential vulnerabilities and countermeasures in cyber security for large organizations. Their domain model allowed them to produce "insider subversion" plans of 40–60 steps, demonstrating the success of applying planning to identify novel attack scenarios. Their work was also the first to identify the problem that planners do not easily produce alternative plans.

2.1.1 The Security Model: A Personalized Attack Graph (PAG)

A natural model for security is a state transition system where nodes indicate states and edges indicate transitions between states. As part of the research grant, we developed a Personalized Attack Graph (PAG) security model to characterize the ways that a *home system* can be compromised [Urb+13]. An important detail of the PAG is that it integrates states of the personal computing system with user and attacker actions. In this model, a vulnerability is some software or system state that is susceptible to attack and user/attacker actions serve as the transitions between states. Vulnerabilities are enabled by specific versions of software or system state and user/attacker actions cause the vulnerability to lead to an exploit. Exploits are then leveraged to attack the compromised system or to access data.

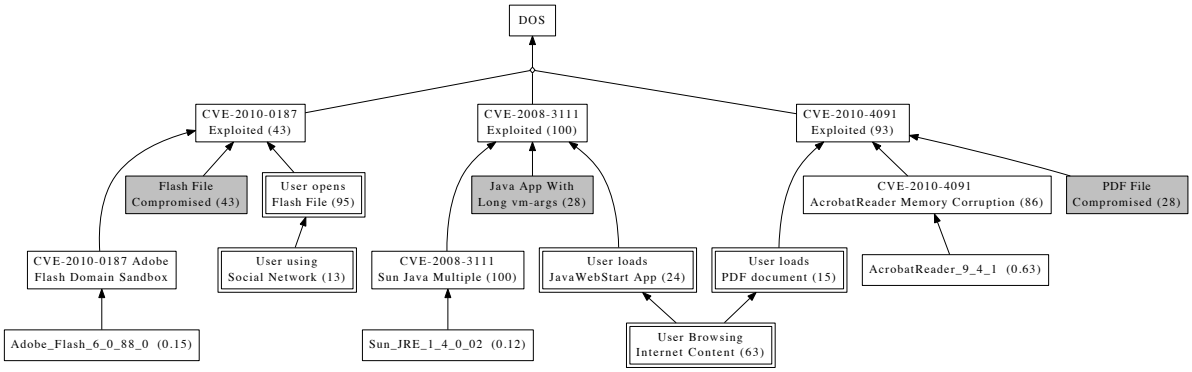

Figure 2.1: The Denial of Service personalized attack graph. The probabilities for nodes are given in parentheses.

Figure 2.1 shows a PAG leading to a Denial of Service (DOS) attack. Urbanska et al. provide a detailed formal description of the PAG [Urb+13], but here we will cover the essential properties relevant to planning. The PAG is a state-transition system that is instantiated with the state of a particular home computer and user. Layers in the graph indicate preconditions, but across the graph the layers are otherwise insignificant. Nodes in the graph include system states and vulnerabilities (shown as white boxes in the figures), execution states of attack actions (gray boxes), and execution states of user actions (doubly-lined boxes). An arc in the graph is used to represent a state transition that contributes to a system compromise, but these transitions are unlabeled for clarity. Conjunctive (AND)

nodes (e.g., the three CVE nodes in the second layer from the top of Figure 2.1) require all preconditions to be met for a state transition; they are indicated by multiple arcs that are incident on the node. Disjunctive (OR) branches (e.g., the Denial of Service (DOS) node at the top of Figure 2.1) have a small circle for the choices and require only a single branch to be true. Nodes that represent the execution of user actions are associated with a user profile [Urb+13]. In Section 4.7, we use this particular example to show how we can generate alternative plans in a single episode

This security model builds on work of a psychology team[1]. The goal of the psychology team is to characterize the user's likelihood of engaging in specific behaviors (e.g., using a social network website, online shopping) and their tolerance for risk. To this end, the team members have performed experiments to determine a user's perceptions of risk and the cost-to-benefit ratio of using Internet technologies [Byr+12] for two populations deemed most at risk from a security standpoint: college-aged adults (aged 18-22 years) and older adults (aged 60 and up). The results from this study are intended to feed into the security model that relies on a user profile to provide probabilities for the user action nodes in the PAG.

A full PAG for a single system can be very large. The DOS subtree of Figure 2.1 is actually only one subtree of a more complete PAG shown in Figure 2.2 that contains 7 exploits, 25 user actions, 38 system states or actions (of which 11 are system vulnerabilities), and 19 attack actions[2]. It was hand-constructed from a subset of vulnerabilities present on an actual machine running Microsoft Windows XP Professional SP3 with common configurations. Before collecting the data, the system was secured and updated. Subsequently, the machine was disconnected from the Internet, and automatic updates were disabled. After three months, the machine was plugged into the Internet and scanned with a vulnerability scanner called NeXpose by Rapid7 LLC [Rap]. NeXpose found 216 vulnerabilities during this scan.

[1]This currently includes Dr. Zinta Byrne and a graduate student, Kyle Sandell.

[2]This PAG was created by Gosia Urbanska.

Of these, 133 were critical, 74 severe, and 9 moderate. In the worst case, the PAG grows exponentially in the number of potential user/system/attacker attributes. For a reasonable size of graph, it would be intractable to represent all the possible ways of an attack occurring. Fortunately, researchers of the more general Attack Graph upon which the PAG is based have tackled this scalability problem.

The PAG is a specialization of a more general model called an attack graph. Researchers have modeled security for networked systems using *attack graphs* [PS98; She+02] and *attack trees* [MEL01; Dew+07]. These models capture dependencies among different system attributes such as vulnerabilities and network connectivity and facilitate security risk analysis and management. But these models focus on networked systems rather than home computer users. Attack Graphs (including the PAG, which is a specialization) quickly become large, computationally expensive to analyze and hard for human analysts to understand. One way to manage this complexity is to make assumptions about how attacks propagate through the graph. One such assumption, monotonicity, states that once a state in the graph is activated, it is never deactivated [AWK02]. The implication is that once an attacker achieves an exploit, the exploit remains active from that point forward. This assumption simplifies propagating changes through the Attack Graph.

Another way to handle the complexity is to use an implicit representation to reason about the graph without instantiating the entire set of states. Ghosh and Ghosh [GG12] reduce the complexity of instantiating an attack graph by iteratively applying a planner to eliminate unreachable attack scenarios. They use a planning model similar to Boddy et al. and generate minimal attack paths. To identify multiple paths that lead to the same scenario, they modified the domain model by eliminating each path (that is, commenting out an action or predicate) as it was discovered. Obes et al. [OSR10] construct a large planning model (1800 actions) from an attack graph and integrate a planner into a penetration testing tool. Although they found an exponential increase in computation time as the number of machines modeled increased, the time was still just 25 seconds to generate a plan involving 480 machines; using planning improves reasoning about an Attack Graph.

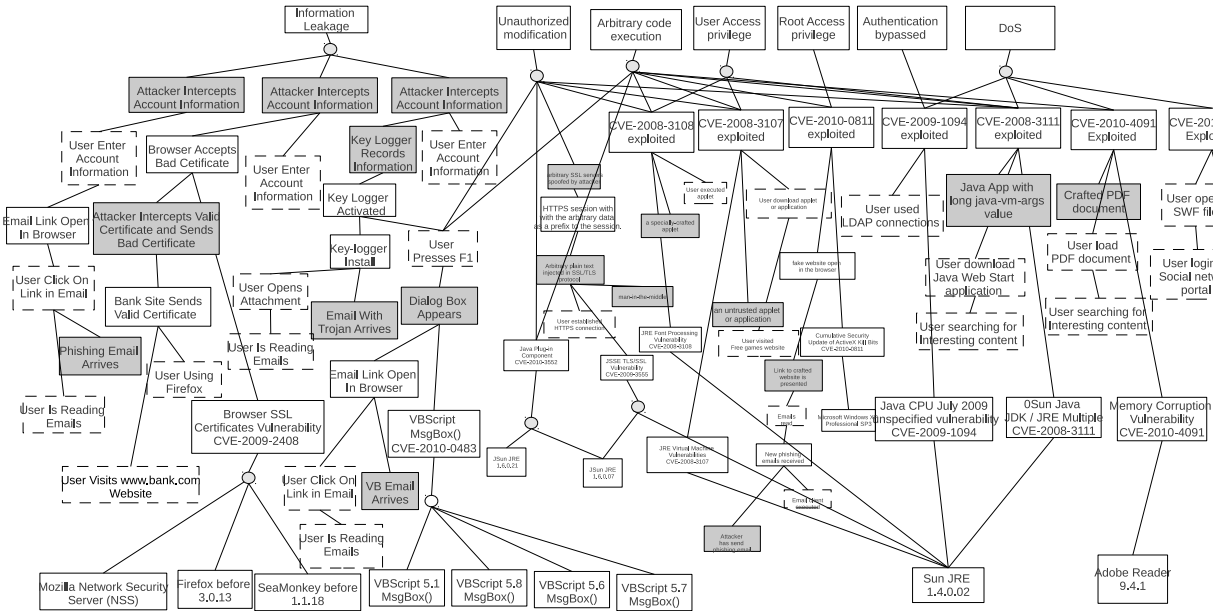

Figure 2.2: Example PAG constructed from 11 vulnerabilities found during a scan of a computer that did not receive patches/updates for three months.

We observe two gaps in the literature for employing planners to solve the computational intractability of reasoning over Attack Graphs and the PAG. In the first case, no previous literature has examined how to enable a planner to find multiple attack paths (i.e., alternative plans) in a single planning episode for these applications. In the second case, these graphs require more than one kind of quality evaluation for the plans that are produced.

2.2 Other Applications of Alternative Plans

The focus problem of a cybersecurity agent for home computer users is only one application that requires alternative solutions or multiple metrics for assessing plan quality. Such concerns are also central in applications for human-in-the-loop control, real-time and remote systems, and decision support.

Many human-in-the-loop AI systems present alternative plans/solutions for evaluation by a human. These systems – which are used in space, military, and commercial applications and sometimes labeled mixed-initiative – can involve a combination of planning and/or scheduling and are often part of a larger software agent. Human in the loop planning in-

cludes applications where humans have final say over plans (e.g., military, computer security) as well as cases where humans have to carry out part of or anticipate consequences of the plan such as some production planning/scheduling and logistics. Example space applications include producing schedules for imaging satellites [SEM02], constructing plans for an autonomous remote satellite [She+98], and scheduling satellite communications [Bar+03]. A plan is constructed and then sent to the remote agent for execution. Due to time lag and safety concerns, plans must meet complex time, resource, and sometimes political constraints. Sometimes the plans must also include contingencies to manage execution anomalies. Thus, planners must support multiple objectives while exploring alternative execution paths.

Real-time applications demand a slightly different focus for generating alternatives because the system generates new plans for a human to assess *during* execution. Applications in this category include managing unmanned aerial vehicles [Gol+02], teams of robots and humans in military campaigns [WLB03], materiel logistics [BS00; SBK04], and an elder's schedule [Pol+03]. These kinds of domains have a strong focus on generating plans that repair execution anomalies but maintain stability with respect to the currently executing plan. Often such reasoning implies understanding the critical bottlenecks. Other real-time applications focus on helping the user identify problems or opportunities in plans. Examples of this kind of system are found in network security [Bod+], vehicle routing [SLK02], and satellite scheduling [Ber+08]. Thus, planners must identify alternatives that may highlight bottlenecks or new opportunities.

For remote, semi-autonomous systems, a plan is constructed and then sent to a system for execution. Often, plans must meet complex time, resource, or political constraints. Such applications include producing schedules for imaging satellites [SEM02], constructing plans for an autonomous remote satellite [She+98], and scheduling satellite communications [Bar+03]. Thus, planners must support multiple objectives.

Decision support applications require human judgment to balance between multiple objectives. Presenting users of these systems with alternative plans gives them options for balancing the objectives. It is often impractical or impossible to capture in a domain model

all the knowledge required to make good judgments, e.g., a resource may be preferred because of which user controls it, or some task should be done earlier because the requesting user has more authority. The problem is exacerbated if there are multiple, possibly competing objective criteria for evaluating a solution. Although many of these systems may employ complex constraint reasoning (e.g., ASPEN [Chi+00] and EUROPA [FJ03]) or hierarchical planning (e.g., PASSAT [Mye+02]), they still rely on human judgment and negotiation. Indeed, in most of the applications, humans are integral to the decision process and *always* approve the final plan. Users must be involved in the decision process when the domain modeling is inaccurate, and planners must support the iterative decision making process of exploring alternatives.

Chapter 3
Background and Related Work

In this chapter, we define formally the classical planning problem and its extensions, explaining the dominant approach for creating plans (heuristic search), discuss recent approaches for generating plan sets, and close with a discussion of how planning systems are evaluated. Our focus is on the systems that take steps along the two research directions of producing plan sets and using alternative metrics. The gray box of Figure 3.1 demonstrates this focus.

3.1 Classical Planning

A common approach to generating plans is to encode the planning problem into a language that a planner can then use to synthesize a plan. The variety of methods for automatically generating plans can fall under the broad framework of "plan refinement" [Sub97], where refinements are added to a (possibly empty or incomplete) plan until the goals are achieved. The three primary means of refining (i.e., searching for) plans are state-based heuristic search, task decomposition, and plan-based search. This dissertation focuses on heuristic search[4], so we begin with a formal definition for heuristic search in planning. We adopt the notation and definitions directly from Gallab, Nau, and Traverso [GNT04]. We assume the state-based planning formalism that is based on a state-transition system. A general state-transition system is a tuple $\Sigma = (S, A, E, \gamma)$, where:

- $S = \{s_1, s_2, ...\}$ is a finite or recursively enumerable set of states;
- $A = \{a_1, a_2, ...\}$ is a finite or recursively enumerable set of actions;
- $E = \{e_1, e_2, ...\}$ is a finite or recursively enumerable set of events; and,
- $\gamma : S \times A \times E \rightarrow 2^S$ is a state-transition function.

[4]In general, when I say *heuristic search*, I mean state-based heuristic search.

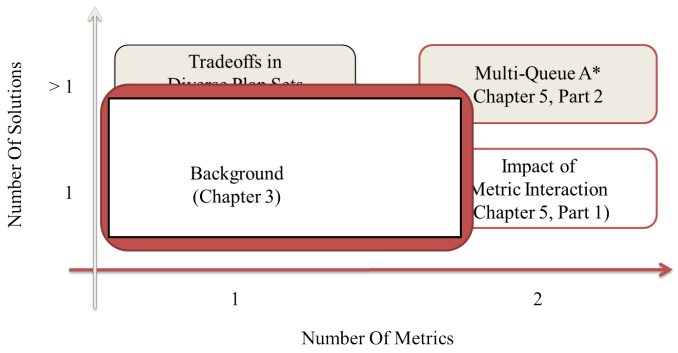

Figure 3.1: The two research directions of this dissertation with the focus of this chapter highlighted in red.

In *classical planning*, it is common to further restrict the state-transition system to 1) be finite, 2) be fully observable (the state of the system is completely known), 3) be deterministic (each transition applies to and results in no more than one state), 4) be static (the set of events is empty and only actions change the state), 5) have restricted goals (the goal or goal set does not have associated trajectory constraints or utilities), 6) have sequential solutions (a plan is a linear ordering of actions), 7) have implicit time (transitions occur instantaneously), and 8) be offline (the world state does not change during a planning episode). The restricted model allows us to disregard the set of events E because those are empty; this system is denoted $\Sigma = (S, A, \gamma)$ rather than the full state-transition system of $\Sigma = (S, A, E, \gamma)$. With this restricted model, planning is then defined:

> Given $\Sigma = (S, A, \gamma)$, an initial state, s_0, and a subset of goal states, S_g, find a sequence of actions $\pi = \langle a_1, a_2, ..., a_k \rangle$ corresponding to a sequence of state transitions $(s_1, s_1, ...s_k)$ such that $s_1 \in \gamma(s_0, a_1), s_2 \in \gamma(s_1, a_2), ..., s_k \in \gamma(s_{k-1}, a_k)$, and $s_k \in S_g$.

A planning problem is then denoted $\mathcal{P} = (\Sigma, s_0, S_g)$, where s_0 is an initial state and S_g is a set of goal states. A solution to this problem synthesizes a plan π that is a sequence of actions. For all but the smallest Σ, it is intractable to enumerate the entire set of states and transitions. A number of representations have been used that can compute the states and transitions as search proceeds. We will focus on the representation that is used in most

planning, called the *Classical Representation*. The de facto language to represent classical planning problems has become the Planning Domain Definition Language [AIP98; FL03], which is a common input language for many recent classical planners.

If cost is assigned to each action, then a cost function $C : A \rightarrow \{0, 1, ...\}$ maps the actions to non-negative integers. In planning, this is called *metric planning*. In cost-based planning, the cost of a plan is the sum of the action costs, or $c(\pi) = \sum_{a \in \pi} C(a)$.

Classical Representations are based on first-order logic that is restricted to be function-free; there can be any number of predicates and constants. A predicate is a relation that gives a truth value; for example, `at(?robot, ?location)`, where `?robot` and `?location` are variables that can be assigned any value from the world.

If a predicate variable is bound to a specific variable, it is said to be *grounded*; otherwise, it remains *lifted*. A constant is a specific object in the system; it can be typed or untyped, though most current systems employ typing to help control the combinatorial explosion due to grounding nonsensical objects to predicates. Parameterized versions of actions, called operators, can also be ungrounded. In this system, an *ungrounded* operator $o \in O$ provides the transition function by listing its preconditions, denoted $\texttt{precond}(o) \in S$, and its effects, denoted $\texttt{effects}(o)$. The task of the planner is to bind variables and select actions. Many planners will fully ground the operators as a preprocessing step prior to search.

Propositional representations only allow binary values for variables. An alternative is to allow an arbitrary number of values for variables. Such multi-valued representations focus on a functional description over a relational description. A commonly used formalism, called SAS$^+$, converts the PDDL problems to a multi-valued representation [BN95]. Note that this formalism can be converted to a propositional representation in polynomial time. The two formalisms are semantically identical, but they can have dramatically diverging computational (i.e., search) costs as the multi-valued encoding can be seen as a more compact set of transitions over which to search.

State-of-the-art planners often fully ground the domain and use a multi-valued representation. The first planner to do so was Fast Downward [Hel06]. This planner employs the

concept of a preferred operators [RH09a], which are identified during successor generation as the set of operators more likely to lead to a goal. A more recent planner, called LAMA [RW10] is based on Fast Downward. LAMA adds the ability to search according to landmarks, which are states in the plan that must be true in any solution to the goal. The idea of a landmark is that it is a kind of bottleneck that can aid searching for a plan.

For the security agent, we can assume a (mostly) classical model. In planning problems where the transitions are non-deterministic, the action effects are stochastic, or states are not observable, it is common to use methods from probabilistic planning. Solutions to probabilistic planning problems are given as policies, which map states to actions. A survey of this area is given by Kaebling, Littman, and Cassandra [KLC98]. Notwithstanding promising advances in state abstraction, a drawback of decision theoretic planning is that the flat state space can be very large even when state abstractions are used. The security agent aligns more closely with an observable, discrete planning model *plus* quality metrics more so than a partially observable, continuous planning model minus a policy.

3.2 Generating Plans

One of the first algorithms for creating plans, A* [FN71], has become a common core algorithm for many planners. A* works by using an open list, $\mathcal{O}$, a closed list, $\mathcal{C}$, and a heuristic function, h. Algorithm 1 shows the A* algorithm. At each step, A* pulls the minimum valued node off the open list, expands its successors (i.e., applies the possible operators that match), evaluates each successor according to h, and places each successor into $\mathcal{O}$. $\mathcal{O}$ is usually a MinQueue (e.g., a MinHeap), sorted according to the function $f = g + \hat{h}$, where g is the cost-so-far of steps to get to this node and $\hat{h} = h(s)$.

We use two techniques that extend A* search. The first uses what is called a *focus list* to select solutions instead of the minimum from $\mathcal{O}$. Instead of getting the minimum state ($s = min(q)$) from the top of the priority queue at each iteration, A^*_ϵ [PK82] selects a solution from the top K solutions. A solution $s' \in K$ if $f(s') <= \epsilon f(s)$, where $\epsilon > 1$ and $h(s') = h(s)$.

Algorithm 1 A*-SEARCH ($\mathcal{P}$, maxSteps) returns a solution or failure

```
1: closed ← ∅
2: O ← INSERT( MAKE-NODE( s_o ), O )
3: stepsTaken ← 0
4: while stepsTaken ≤ maxSteps do
5:    if isEmpty( O ) then
6:       return failure
7:    end if
8:    node ← O.removeNext()
9:    state ← node.getState()
10:   if S_g ⊆ state then
11:      return SOLUTION( node )
12:   end if
13:   if not inClosedList( state ) then
14:      closed ← closed ∪ state
15:      O ← O ∪ EXPAND( node, P )
16:   end if
17:   stepsTaken++
18: end while
```

An important property of A^*_ϵ is that, assuming an admissible heuristic, it can guarantee the solution quality is within $(1 + \epsilon)c*$, where $c*$ is the optimal solution quality. The second technique is modifying the sort of $\mathcal{O}$ using $f = g + wh$, where $w > 1$. This technique favors the heuristic during search so that the search process is biased to explore states closer to the goal before expanding states farther away from the goal.

3.2.1 The Plan Graph and Modern Heuristics for Classical Planning

An important advancement in state-space heuristic search for planning was the Planning Graph [BF95]. This data structure interleaves the states and actions of the state-transition system. Figure 3.2 shows the first three layers of the graph for the problem in Figure 1.1. The first state level is the initial state of the system, and the first action level contains those actions that apply to that initial state. The next state level is given by the effects of each action that can apply, and mutually exclusive (mutex) relations (not shown in the figure) are put in the level to account for the case where a state contains contradictory effects. A no-op action carries all effects from the previous level that do not have a corresponding action. States accumulate as the levels increase provided that more actions can apply at each new

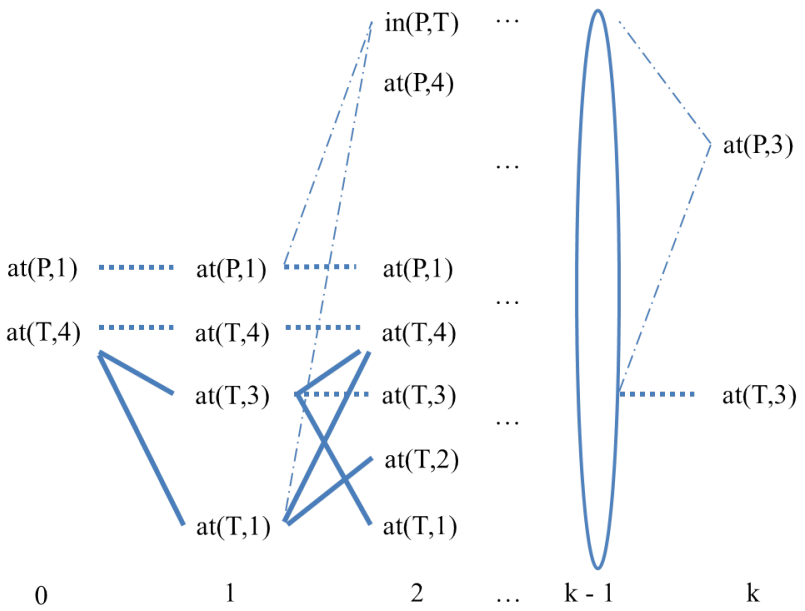

Figure 3.2: The first three layers of the plan graph (without mutexes) for Figure 1.1. Solid lines represent the move operator, circle-dash lines represent special no-op actions that transfer unchanged state from one layer to the next, and long-short-dash lines represent load/unload for the package.

level. When a level is reached that contains the goal state(s), as shown by the $k-1$ and k layers in the figure, then one can search the graph to recover a solution.

Few planners continue to use the planning graph at their core, but a more recent application of the planning graph is in generating heuristics to use during heuristic search. The planning graph is useful for providing a lower-bound on the number of steps to achieve some state or goal condition. Since delete effects generate most of the mutexes for the construction of a planning graph, ignoring them considerably speeds up computation. So a relaxed version of the planning graph is used wherein the delete effects are ignored. This heuristic is commonly referred to as h^+. The first planner to use this heuristic was the Heuristic Search Planner (HSP) [BLG97; BG01]. The h^+ heuristic, or one of its family of variants [HG00; HBG05], are now widely used in heuristic search planning.

3.2.2 Other Methods for Generating Plans

While the focus of this thesis is in extending the state-of-the-art approaches in state-based heuristic search, we cover additional methods for planning that could make sense for some aspects of security agent but were not studied.

Precompiled Plans A precompiled (or hand-generated) plan could be used for activities that the agent should know how to accomplish without additional reasoning. For example, we already know how to update software, so it makes more sense to create a precompiled plan that will work for most software updates. A possible list of precompiled plans includes: updating software for major programs, updating the OS, updating virus definitions, checking/confirming system registry and user access settings, monitoring actions, system actions, scheduling and queuing actions, and checking for newly installed software.

Plan Templates In many cases, a plan library can also provide plans that have either been hand-generated or previously discovered through automatic generation. Not all domains require automatic plan generation and storing a set of previously generated plans can "free" the agent's resources to focus on other concerns. In some domains, it makes more sense to have domain experts encode possible plans and then have a planner place these plans on a timeline. The security agent could use hand-generated plans when they are appropriate, as in the case for intervening in security attacks.

As an example of manually generated templates, plans for Autominder [Pol+03; Pol02] consisted of daily plans constructed by a caregiver on behalf of, or in consultation with, an elder client. The caregiver could select from a set of *plan fragments*, for daily activities such as "Watch a TV program" or "Take a medication." The caregiver would fill out the template with more specific details regarding recurrence and preferences such as "three times daily" or "within an hour after breakfast." Daily plans consist of a set of completely specified plan fragments. The plans are represented in a constraint-based representation called a Disjunctive Temporal Problems (DTPs). DTPs can leverage constraint satisfaction techniques to quickly compute whether the current plan is feasible given all the constraints

and where potential bottlenecks exist. The DTP model also allows for easy online updates to the plan. Updates occur when activities are added or removed from the plan, when an activity is completed, or when an activity is not completed.

Task Decomposition In a Hierarchical Task Network (HTN) [EHN94b; EHN94c; Ero+95], the planner decomposes goal states into methods and task networks that further refine a plan. As the planner applies possible expansions of methods and tasks, it selects among choices to achieve goals. This approach is useful for combining automated search with plan templates. HTN planning is exemplified in the SHOP2 planning system [Nau+03]. Although HTN planning has the same complexity as other planning techniques [EHN94a], its key advantage is in domain elicitation and domain construction, where the domain engineer can specify how to achieve a goal by decomposing the goal into sets of possible expansions.

HTN planning is applied at the highest reasoning level of Phoenix, a multi-agent system for forest fire management [Coh+89; How93]. The top-level agent, called a fireboss, plans through the use of a plan library and a hierarchical decomposition. Three action types can be scheduled on a time-line to allow this decomposition to take place in a natural way. *Selection actions* are place holders on the time-line that tell the agent to search for appropriate plans in the plan library for a specific goal such as "deal with a new fire." *Plan actions* instantiate (or expand, to use HTN language) the resulting plan that was chosen by the selection action. *Primitive actions* direct an agent or command a sensor or effects. All three types can be composed of each other, which allows for nested plan expansions.

3.3 Generating Alternative Plans

Alternative plans capture the variety of ways that a set of goals can be achieved. They can be used to increase applicability of planning systems to support human decision making and agent adaptability. For example, in mixed-initiative planning, alternatives can explain key bottlenecks and represent a variety of solutions [SLK02]. In an execution setting, new plans may deal with contingencies while minimizing disruption to an already executing plan

[Fox+06]. In a security application, the planner needs to produce all the possible ways an attack can occur [Bod+].

Several groups of researchers have examined how one can generate alternative plans that are diverse with respect to plans that are already found. Approaches for generating alternatives can be broadly characterized along three dimensions: 1) whether they modify a single solution or a population of solutions, 2) whether they use a full planning system or single algorithm, and 3) whether they focus on changing the heuristic or controlling the set of states visited during search.

Several groups of researchers have examined how one can generate alternative plans that are diverse with respect to plans that are already found. Srivastava et al. [Sri+07] explore how to generate diverse plans in a constraint-based planner and a local search planner; this work was later extended to planning with incomplete preference models [Ngu+12]. They use actions, states, and causal links to assess the distance between plans, and they use a weighted combination of one of these distance metrics during search to produce diverse plans. They find that using actions usually encourages sufficient search diversity.

Coman and Muñoz-Avila [CMA11] describe a heuristic-based approach that can embed arbitrary distance metrics into the search heuristic. Given a plan π, a set of plans Π, and a distance metric D, this method calculates:

$$\text{RelativeDiversity}(\pi, \Pi) = \frac{\sum_{\pi,\pi' \in \Pi} D(\pi, \pi')}{|\Pi|} \tag{3.1}$$

Generalizing the distance metric, D, in Equation 3.1 allows the authors to substitute any quantitative or qualitative distance measure to help generate diverse plans. In particular, as the authors point out and show, D can contain domain-specific information that may be challenging to incorporate into the domain model. For example, the authors use D to guide planning in a Real Time Strategy game where the domain model remained the same but D took into account dynamic environmental concerns that are absent from the domain model. They showed that they could elicit different plans from the planner that adapted to this new information without directly changing the domain model.

More recently, Talamadupula et al. looked at generating plans for execution with the best net benefit in a partial satisfaction planning framework [Tal+10; Sch+09]. While this work is not strictly about generating alternatives, it does examine how to evaluate multiple plans with respect to usefulness to a user. For their evaluation metric, they use the sum of the reward minus the sum of the action costs.

We note that many of the planners from the recent International Planning Competition (IPC-2011) find better solutions over time, e.g., LAMA [RW10], which uses a multi-queue local WA* search, and CBP [Fue11], which uses branch-and-bound search. Such anytime planning algorithms and satisfying planners could be seen as generating alternative solutions; each new solution is effectively an alternative that improves over the last. Indeed, nearly any planner could be modified to produce alternative solutions, though they may still have a bias toward progressively better solutions under the current IPC metrics for comparing planners.

3.3.1 Multi-objective Planning

Temporal and resource reasoning lies at the heart of many significant problems in planning research. A wealth of literature exists for producing (diverse) temporal plans using a single plan metric (e.g., SAPA [DK03], LPG-td [GSS06], COLIN [Col+12]). In most cases, these temporal+metric planners embed deep reasoning to solve specific resource and time constraints that lie at the junction of planning and scheduling. It is difficult to assess the contribution of the non-temporal metrics when they are combined with temporal concerns in a weighted objective function.

Several researchers have examined multiobjective planning in different contexts such as temporal planning or preference planning. The SAPA planner by Do and Kambhampati [DK03] is a multi-objective metric temporal planner that can handle resource constraints; this was one of the first planners to examine how to extend the planning graph heuristics to incorporate plan quality and plan makespan. Nguyen et al. examined multi-objective planning for problems where the preferences model is incomplete [Ngu+12]. As

mentioned in Section 3.3, they used actions, states, and causal links to encourage search for diverse solutions. From a multi-objective perspective, they evaluate the planners according to a metric called the Integrated Convex Preference [Fow+05], which is a way to assess the solution quality of a set of solutions in a weighted multiple-objective context. More recently, Khouadjia et al. [Kho+13] wrapped a population-based search mechanism around a planner, YAHSP [Vid04], to produce Divide and Evolve YAHSP (DAE-YAHSP)[5]. This planner produces diverse solutions and handles multiple quality metrics.

3.4 Evaluating Planning Systems

In general, the planning literature has focused on producing a single solution while minimizing (or maximizing) a single metric – metrics in the planning literature are known by objective functions or evaluation criteria in other search literature. There are notable exceptions such as the planners mentioned in the previous two sections. Many applications require complex judgments that seek a *balance* between a variety of objectives and not just progressively better solutions driven by a single metric. We now cover the metrics used in our study

3.4.1 Efficiency

A standard way to evaluate software systems is by the computational resources they consume during their execution lifetime. The field of planning uses similar CPU time and memory measures as those in much of computer science. Since planning systems are large software projects, an upper bound on computational resources is usually set (e.g., 15 minutes and 1 GB of memory). When two approaches share the same underlying code base, they can also be compared by the number of nodes they generate or the number of evaluations of the heuristic function. Both are proxies for the true effort but provide a fair comparison.

[5]We attempted to obtain a copy of DAE-YAHSP for our study, but were informed that the authors are creating a version for public release.

3.4.2 Quality

In many applications, humans (or agents) must balance complex, hard-to-quantify objectives. The quality of a plan is useful in the context of alternative plans because it helps the human (or agent) evaluate the various means of achieving goals. We consolidate under the term of "plan quality" a number of measures that can be quantified, which might include: the utility to the user, the number of actions in a plan, the likelihood of occurrence for a plan, the plan's temporal duration (i.e., its scheduling makespan), the resource usage of the plan, etc. Several of these metrics come from the security domain that motivates our research, while others appear in the planning competitions that we discuss next.

For a long time, the classical planning literature – at least the literature outside of applying planning to realistic problem domains – focused on minimizing the number of actions, also called the plan length, of a serial plan. Applications that required more reactive planning (and the use of decision theoretic models such as MDPs or POMDPs) have included quality measures and leveraged decision theory to come up with policies that maximized the expected gain [KLM96]. Similarly, applications where scheduling concerns played a central role [Fox94; BS00] included many evaluations of the overall quality of a plan/schedule[6]. We quote Bacchus, the organizer of the second International Planning Competition (IPC2): "In general, no serious attempt was made to measure the quality of the plans produced. Only the length of the plan was measured because there was no facility for specifying the cost of different solutions. In fact, most of the planning systems were incapable of taking such a metric into account during planning." [Bac01, p. 49]

[6]It is especially noteworthy to see the overlap of scheduling and Partial Order Planning (POP) [Wel94; SFJ00] and their application to real-world applications, which continue to be solved using partially ordered planning in some combination with HTN and/or timeline-based systems. The IPCs are just coming around to evaluation metrics that are getting closer to the needs of those real-world applications. But state-based single-metric and single-solution planning systems still dominate the competition entries.

Plan quality in the International Planning Competitions In the last ten years, interest in quality-aware planning has surged, and, since IPC2, the role of quality metrics in evaluating plans has gained substantial footing. The IPCs provide a overview of that change in focus. Most notably, the designers of the IPCs have progressively expanded the representational ability of PDDL, the evaluation of planners has begun to include these metrics in greater earnest, and planners have slowly adopted the comparison of plans under various quality metrics.

The 2002 competition, IPC3, added to PDDL durative actions, a clarifying of numeric support, and support for arbitrary plan metrics [FL03]; this version used a subset of PDDL+ [FL06] and became known as PDDL 2.1. Most of the problems in the competition had 4 metrics: plan length (called `strips` by the organizers), sequential makespan (called Simple-Time), concurrent makespan (Time), and a metric based on the domain (Numeric). Two additional problem sets had an optimization metric that measured the quality of images taken by a satellite (HardNumeric), and an optimization metric that combined satellite resource usage with the solution makespan and the image quality (Complex).

The organizers, Maria Fox and Derek Long, spent considerable effort understanding how planners compared with respect to the plan metrics [LF03]. Of the fully-automated planners, very few planners attempted the so-called HardNumeric and Complex problems. Similarly only three fully-automated planners completed the numeric domains. Across the four metrics, the results demonstrated that it was challenging to compare planners in terms of plan quality, but that the inter-planner rankings were stable. Only a small set of the planners used metrics other than temporal duration. One of the truly surprising results of the competition analysis was that hand-coded planners did not seem to gain a huge advantage over the domain independent planners in terms of the quality metrics, although the authors do point out the limitations in such a conclusion because of the post-hoc analyses.

The fourth competition in 2004, IPC4, sought to further extend the focus on quality metrics. The organizers, Jörg Hoffmann and Stefan Edelkamp, added derived predicates and timed initial literals to the language [HE05] and the resulting language version was

PDDL 2.2. Derived predicates add a kind of domain axiom that becomes true as a result of non-planner actions and are a convenient way to express completion of paths or flows. Timed initial literals add what has become more commonly called exogenous events, which are a convenient way to represent time windows for specific events.

As with IPC3, the organizers of IPC4 spent considerable time analyzing the data. No hand-coded planners competed in this competition. The deterministic track of the competition was further split into optimal and satisficing planners; there was also a probabilistic track that will not be discussed here. The predominant measures of plan quality were scheduling makespan and plan length. Only one domain, Satellite, used a quality metric that diverged from plan makespan or scheduling makespan. The metric of the `Numeric` version was to minimize the fuel usage, and four of the five planners attempted to take this into account during planning. The `Complex` version used a metric that was a linear combination of the makespan, the fuel usage, and negative image quality, but none of the five planners that could solve this problem attempted to use the metric during planning.

IPC5, held in 2006, was organized by Alfonso Gerevini, Yannis Dimopoulos, Patrik Haslum, and Alessandro Saetti. The competition featured PDDL 3.0, which extended PDDL 2.2 to include state trajectory constraints that are true over every timepoint of the constraint window, and preferences over constraints and goals [Ger+09]. The competition featured numerous domains with domain-specific quality metrics in addition to metrics such as plan makespan, soft-goals and preferences. Of the seven satisficing planners, five used the problem metric during search, though some of the planners used the metric in limited situations (i.e., SGPlan5 [CWH06]). Only two planners supported the full set of PDDL 3.0 and only one additional planner supported the full subset of PDDL 2.2. Few planners could deal the more complex PDDL syntax and the more complex problems. For quality, many solutions were much worse than the best/optimal known solutions. For preferences, the planners were finding better solutions than when ignoring preferences. The organizers posit that this is because the planners are able to find better solutions because of the techniques used during search [Ger+09].

IPC6 was held in 2008 and IPC7 was held in 2011. Unfortunately, no peer-reviewed articles have been published that summarizing them, so the conclusions that can be drawn are limited to the brief presentations given at the conference venue. Neither competition extended PDDL in a significant way except to correct or limit some semantic concerns, which suggests that the language was more complex than many planners could handle. IPC6 held a net-benefit track that used the full PDDL 3.0 capability. However only three optimal planners competed and the satisficing track was canceled due to lack of participation. The conclusions that can be drawn from the presentation of the net-benefit optimization track are very limited. IPC7 had no "metric" or "net-benefit" tracks and chose instead to focus exclusively on strips and temporal domains with action cost or plan makespan being the key metric.

Non-competition Approaches to Incorporating Plan Quality. In parallel with the competitions, other researchers have examined plan quality. Schreckenghost et al. built a mixed-initiative planning assistant that helps a user examine possible plans/schedules for a space crew planning problem [Sch+03]. The authors provide the user with several metrics to evaluate the plans: goal subset selection (why specific goals, among many, were chosen for planning), goal achievement, compliance with user preferences, goal priorities, plan efficiency, and resource usage. Plans are ranked by an ordering of these metrics as specified by the user. The system also supports managing/storing alternative plans as well as giving the user a way to capture them. The planning horizon is managed through the notion of duty blocks, where only the relevant available time slots are considered for planning. During plan construction, the planner greedily selects tasks based on the user's supplied metrics. During execution, replanning is used to handle execution anomalies (though this is a proposed component and not part of what was actually implemented).

Do and Kambhampati [DK04] extended the traditional relaxed plan heuristics of the SAPA planner [DK03] to include action cost using net-benefit planning, which attempts to maximize the utility (i.e., the net benefit) of a plan while minimizing the action cost. Benton,

Do, and Kambhampati [BDK09] then applied this cost-based extension in oversubscribed planning domains. These domains often contain more goals than can be satisfied, so the objective is to meet as many high utility goals as possible. The authors present a Partial Satisfaction Planning (PSP) framework to handle these oversubscribed planning problems that have goal utilities and action costs.

There are two significant contributions of the research in PSP. The first is a set of specific heuristics that deal with cost during search. The heuristics allow the second contribution that soft-goals are kept as an integral part of the search, which allows for a direct reasoning about the trade-off of utility versus action cost *while* searching for a plan. In contrast to an approach that compiles away soft goals in a preprocessing step [KG09], the authors conclude that it was more beneficial to maintain the original formulation of soft goals or to compile them away into hard goals. They found that compiling preferences to hard goals reduced search effectiveness.

Keyder examined a number of extensions to the h^+ heuristics that incorporate action costs [KG08; Key10]. The h^+ heuristic over-estimates the actual cost of achieving a plan because it is the sum of the subgoals; it is a pessimistic upper bound on the true cost. On the other end, an admissible version of h^+, called h^{max}, takes the maximum value of the subgoals; it is an optimistic lower bound on the true cost. Keyder and Geffner introduced a more accurate heuristic, called the set-additive heuristic h^{sa} that counts the subsets of actions that appear in the subgoals. This heuristic finds better quality plans than h^{max} and another common h^+ heuristic called h^{ff}, although the authors point out that the set-additive heuristic is often an order of magnitude slower than the other two heuristics [Key10].

A criticism of the net-benefit or PSP approaches is that they maximize the quality metric of interest after subtracting the plan cost. This can have multiple drawbacks. If action costs swamp the quality, then search is biased toward the action cost space. If two plans have wildly different action costs *and* different quality metrics, then you must select one over the other. Recent research shows that cost-based search – that is, search driven by the metric rather than the plan length – is sensitive to a number of issues. Cushing, Benton, and

Kambhampati [CBK11] show specific cases where cost-based search is easily misled to find a short plan of very poor quality when a longer plan of much better quality exists. Wilt and Ruml [WR11] demonstrate that search is sensitive to the ratio of operator costs. Sroka and Long [SL12] assess the metric sensitivity of planners and show that MetricFF (and other planners) can generate more diverse solutions by varying the constraintedness of resources in a logistics domain. What is not known from these results is the extent to which such pathological behavior is elicited impacted when multiple quality (and cost) metrics interact with each other. Further, it is unclear whether incorporating diversity as a mechanism to drive search further confounds such pathological search behavior.

3.4.3 Diversity

A common way to evaluate the diversity within a plan set is to use some distance metric, D, between the plans within that set. One distance metric is to count the difference in terms of the number of ground actions. Fox et al. [Fox+06] codify this measure in research where the algorithm tries to generate plans that are stable with respect to an already executing plan. The authors study an online execution context, where plan stability is important because large changes in the plan could result in surprise, irritation, or wasted execution effort. So they define a stability metric such that if π_1 and π_2 are the actions for two plans, then,

$$\mathrm{D}_{stability} = |(\pi_1 \setminus \pi_2)| + |(\pi_2 \setminus \pi_1)|. \tag{3.2}$$

This equation is intended as a distance metric between the actions of two plans. Two plans that are similar are presumed to have similar action sequences. Coman and Muñoz-Avila generalize this measure to a plan set, Π, as follows:

$$\mathrm{Diversity}_{\mathrm{stability}}(\Pi) = \mathrm{D}_s(\Pi) = \frac{\sum_{\pi,\pi' \in \Pi} \mathrm{D}_{\mathrm{stability}}(\pi, \pi')}{\frac{|\Pi| \times (|\Pi|-1)}{2}}. \tag{3.3}$$

Srivastava et al. [Sri+07] (later extended by Nguyen et al. [Ngu+12]) explored actions, execution states, and causal links to assess differences in plan diversity. They find that using

an action-based distance usually generates diverse plans compared to the state and causal-link distance measures. So we focus on the action-based distance. The distance formula,

$$\delta_a(\pi_1, \pi_2) = 1 - \frac{|(\pi_1 \cap \pi_2)|}{|(\pi_1 \cup \pi_2)|} = \frac{|(\pi_1 \setminus \pi_2)|}{|(\pi_1 \cup \pi_2)|} + \frac{|(\pi_2 \setminus \pi_1)|}{|(\pi_1 \cup \pi_2)|}, \tag{3.4}$$

is a plan-length normalized version of Equation 3.3. Although the authors only applied δ_a during search, we use it to calculate a normalized diversity metric by applying it in the same way as Equation 3.3,

$$\text{Diversity}_{\text{norm}}(\Pi) = \text{D}_n(\Pi) = \frac{\sum_{\pi,\pi' \in \Pi} \delta_a(\pi, \pi')}{\frac{|\Pi| \times (|\Pi|-1)}{2}}. \tag{3.5}$$

A drawback of distance-based metrics is that they can overlook other ways in which the plans are similar. For example, the distance metrics fail to examine how the plans within a set may actually be repeated, permuted, or subsumed by other plans. Further, it is not clear from the distance-based metrics how to compare plan sets to each other. We address these shortcomings in Section 4.1.

Chapter 4
Assessing Tradeoffs in Generating Diverse Plan Sets[7]

A planner's performance is typically evaluated along several dimensions for producing a plan π: its *coverage* over the number of problems it solved, the computational *efficiency* of producing each plan, and the *quality* of its final plan for each problem. A challenge for comparing planner efficiency is that planners are often sophisticated software systems that elude direct and detailed complexity analysis. When two planners share the same implementation (or two techniques exist within the same planner implementation), then efficiency can be compared using the number of search nodes generated, which is often called the search cost and is a proxy for time and memory. When planners do not share the same implementation, total CPU time to a valid plan is reported within upper bounds on the available computational resources (e.g., 30 minutes and 1 Gigabyte of memory).

Along the quality dimension, planners are usually assessed by one of two metrics: plan length or plan cost. *Plan length* denotes the size of the plan, $|\pi|$, or the number of actions in the plan. In temporal planning, plan length is an upper bound on the plan makespan, which denotes the number of steps required to execute the plan and can include cases where actions might execute in parallel. When plan quality is evaluated using plan length, an assumption is that shorter plans are preferred to longer plans. Plan length as a quality metric is deeply embedded in many planning systems regardless of whether shorter plans are appropriate for the problem being solved. For example, common algorithms for planning (e..g., best first search, see Section 3.2) rely on reachability analysis that usually returns the "closest" plan to the current partial solution (see background Section 3.2.1).

Plan cost denotes the cost of executing the actions of the plan, $c(\pi)$, which assumes that each action, $a \in \pi$, has an associated cost $c(a)$, and $c(\pi) = \sum_{a \in \pi} c(a)$. When plan quality is

[7]Portions of this work are from the paper: M. Roberts, A. Howe, I. Ray, M. Urbanska. *Using Planning for a Personalized Security Agent.* The AAAI-12 Workshop on Problem Solving using Classical Planners (CP4PS-12), July 22, 2012, Toronto, Ontario, Canada.

evaluated using plan cost, an implicit assumption is that lower cost plans are better. At first glance, using plan cost to drive the search has the appearance of moving away from plan length. Yet many planners still rely on reachability analysis as the basic mechanism driving search. Plan quality even has the undesirable property that it can severely mislead search (see background Section 3.4.2).

Coverage, quality and efficiency have a long history of use in the planning literature for comparing planners that produce a single "best" solution. Generally speaking, a more efficient planner will solve more problems and have higher coverage. So, in this evaluative context, one planner is preferred over another planner for a given problem if it produces a better quality solution with the same (or less than a set bound of) computational resources. This evaluative focus has naturally led the community to focus on the tradeoff between efficiency and quality, and has led to techniques that improve performance along one or both of the quality-efficiency dimensions. The end result is continuing advancement of the state-of-the-art for applications needing the best quality solution. In general, planners get faster at finding that best solution.

But, as we motivated in Chapter 2, some applications require the planner to produce a *plan set*, Π, instead of just one solution. This relaxation to alternative plans admits another dimension of evaluation, namely the plan set *diversity*, which characterizes the way in which plans within the set differ from each other. To be sure, it is still desirable to maintain suitable

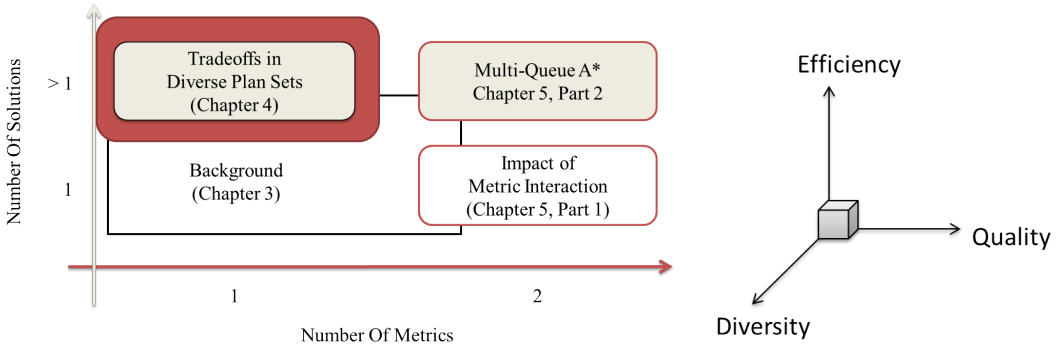

Figure 4.1: On the left, the two research directions of this dissertation with the focus of this chapter highlighted in red. On the right, the three main axes of evaluation we consider.

quality-efficiency tradeoffs. In this revised evaluative context, one planner is preferred over another if it produces plan sets that are highly diverse while still maintaining good quality and reasonable efficiency. Although recent work does examine how to incorporate diversity metrics into planners [Ngu+12; CMA11; Sri+07], relatively little work examines how using diversity metrics to drive search and evaluate planner performance may confound analysis and may guide search to produce solutions that fail to match the desired outcome. We hope to tease apart the two uses in our work.

In this chapter, we investigate the tradeoffs of moving in a research direction of producing plan sets by evaluating several approaches according to the axes of efficiency, quality, and diversity. Figure 4.1 demonstrates this focus. Our analysis leverages existing metrics for evaluating efficiency and quality by using the common measurements already mentioned: search cost, coverage, plan length, and plan quality. Along the diversity axis, we evaluate plan sets using two diversity metrics from the literature [Ngu+12; CMA11]. We are interested in the interplay of diversity-efficiency axes, so we introduce two new evaluation metrics: the *uniqueness* attempts to characterize how much duplicated effort exists within a plan set, while a set-based metric, *overlap*, helps compare two approaches to each other. Finally, we investigate how the plan set changes over time to assess whether additional effort is worthwhile.

Our study includes four algorithms and one full planning system. To maintain a fair comparison, we implement the algorithms in the same planning framework. The algorithms include random walk search [XNM12], modifying the heuristic to drive search [CMA11], and modifying the states visited using a Tabu list [Rob+12], plus a hybrid of the heuristic and Tabu approaches. We complement our analysis of the algorithms with a full planning system, called LPG-diffmax [Ngu+12], that is designed to produce diverse plan sets. We run these five approaches on several benchmark domains from the International Planning Competitions (IPCs).

Our results show that, although the full planning system solves more problems, it does not always produce more diverse plans than the simpler algorithmic approaches when we

take into account our new evaluation metrics. We show that adding diversity to a planner's search mechanism often leads to many duplicated solutions or to solutions that contain duplicated action sequences; i.e., the plans are syntactically different but uninteresting alternatives because they are subsumed by another (valid) plan. Paradoxically, the approaches that generate the highest diversity with the highest uniqueness also produce plans that include many extra actions that sacrifice plan quality and reduce search efficiency. Thus, we show that long searches for more diverse plans are unnecessary with these approaches. Our findings suggest that selecting the best approach for generating alternatives depends on the evaluating metric(s), which ultimately depends on the needs of the application for which those alternatives are generated. Most importantly, our findings challenge the approach of using diversity as a metric to both drive search and evaluate planner performance.

4.1 Diversity Metrics

We examine two ways to assess plan sets. The first compares the diversity *within* a plan set, while the second is to compare plan sets *between* two sets generated by distinct approaches. Within a plan set, we apply two existing diversity metrics in our work. The first is due to Coman and Muñoz-Avila [CMA11] (see Equation 3.3) the second is due to Srivastava et al. [Sri+07] (see Equation 3.5, which was later also used by Nguyen et al. [Ngu+12]. Both metrics are based, in part, on the observation by Fox et al. [Fox+06] that the actions are a natural way to establish distance between plans.

These metrics summarize the average distance within a plan set. However, these values can easily be misleading when plans are subsumed or permuted versions of each other. These distance metrics also have some shortcomings. First, using distance metrics to drive search and also evaluate the outcome leads to a confounding factor. Second, it is not clear how to transfer the distance metrics for comparing between plan sets of different approaches. Finally, it is also unclear how distinct plans are within a plan set because the distance metric obscures this in a single value. So we contribute three additional diversity metrics designed to overcome these limitations: uniqueness, parsimony, and overlap.

Some plans pad a shorter plan with spurious actions while other plans permute the actions of another plan. We capture this in a measure called **uniqueness**, $u(\Pi)$ that reduces the plan set to those plans that are not subsumed after removing padded and permuted plans.

$$u(\Pi) = \sum_{\pi_m, \pi_n \in \Pi, \pi_m \neq \pi_n} \begin{cases} 0, & \text{if } \pi_m \setminus \pi_n = \emptyset \\ 0, & \text{if } \pi_n \subset \pi_m \\ 1, & \text{otherwise.} \end{cases} \tag{4.1}$$

Note that uniqueness is a measure of the efficiency with which the planner finds novel plans. Low uniqueness usually indicates wasted search effort, although we show later how this measure can be misleading because it ignores action sequences that don't lead to a goal. We occasionally limit a plan set to the first i unique plans to maintain a balanced comparison between approaches. Often, we use the first $i = 10$ unique plans because it ensures a fair comparison to approaches that did not produce many plans.

The uniqueness metric does not capture all cases where a plan contains extraneous actions that do not produce a useful goal-directed alternative. The original plan, π_l, contains a subset of k actions that still lead to a valid plan. Thus, the plan is not as parsimonious as it could be. Although such plans can be highly diverse from the others in the plan set, we will show that they often lie at the worst extreme of the efficiency-quality axes that the planning community cares deeply about; these plans are neither efficiently produced nor of good quality. Driving search to maximize diversity can cause a planner that normally produces concise plans to sometimes produce longer plans of poorer quality with significant duplicated search effort. Or worse, the planner may produce plans including action sequences that are unrelated to the goal state.

Characterizing the extent to which a plan might contain spurious action sequences is challenging. To find a plan of minimal length, π_k, is as hard as planning itself in the worst case. But let us suppose that this analytical worst case is not hit often enough that it prohibits good empirical results. Suppose we have a given plan $\pi_l \in \Pi$ with length $l = |\pi_l|$. We want to compute a minimal plan, $\pi_k \subseteq \pi_l$ of length $k = |\text{minimal}(\pi_l)|$ that includes only the actions necessary to produce a valid minimal plan. A naive approach to find π_k is

through ablation of π_l, which may have the same worst case complexity as planning itself[8]. A different approach leverages best first search to construct π_k. For our metric, we search for π_k using A* search with only the actions from π_l. Using only the actions from π_l significantly reduces the branching factor of the search space. Note that there could be more than one minimal plan, but no plan will be shorter because A* is optimal in this setting (i.e., when searching by plan length). Note also that a solution may not be found if l or k are sufficiently large or if A* is not given sufficient computational resources.

With π_k at hand, we can *estimate* how often our approaches violate parsimony. We calculate a **parsimony ratio** to capture how well a planner generates the smallest valid plan with a given set of operators. To obtain this ratio for each plan, π_l, we run A* search with only the actions from π_l to produce a minimal plan π_k. However, given the number of solutions we need to process, we run each solution with a reduced computational bound of 2 hours and 1GB of memory. So it is possible that search will not find a minimal plan. For the plans where A* produces a minimal solution, π_k, we calculate the parsimony ratio,

$$s(\pi_k, \pi_l) = |\pi_k| \setminus |\pi_l|, \tag{4.2}$$

where π_l is the original plan and $\pi_k \subseteq \pi_l$. The variable s is chosen because it does not conflict with the typical usage of p in statistics or probability contexts. The ratio s demonstrates on a scale of $[0, 1)$ how close in length $|\pi_l|$ is to $|\pi_k|$. Higher values of s are better as they indicate that the planner finds an alternative plan close to the minimal length *given the operators from* π_l. We report summary statistics of this ratio over a plan set.

Finally, when considering how plans sets *between* algorithms compare, we assess the **overlap** as the set intersection of two plan sets,

$$o(\Pi_1, \Pi_2) = \Pi_1 \cap \Pi_2. \tag{4.3}$$

Plans can be distinct from each other while still being supersets of the unique plans.

---

[8]This is conjecture on our part, and parametrized complexity analysis may show a naive approach is tractable for small values of k and l. However, n-way interactions may prove to be the bane of this approach.

4.2 Domains

We study six domains. The first is the security domain discussed in Section 2.1. The other five are benchmarks from the International Planning Competition (IPC) as shown in Table 4.1; this table also shows the count of problems, actions, and predicates for each domain (or each problem in the case of `Cybersec`) to give a sense of the size of these planning problems. Three IPC-2002 domains were used by Coman and Muñoz-Avila [CMA11]: `Driverlog`, `Depot`, and `Rover`. We add the Cyber-security (`Cybersec`) domain from IPC-2008 inspired by the work of Boddy et al. [Bod+] that identified the issue of generating alternatives. Finally, we include the seq-opt `Transport` domain from IPC-2011 because it is a newer "logistics" style benchmark, makes sense in a mixed-initiative setting, and includes action costs that we can use to examine plan quality in later experiments. We use the `Transport` problems from seq-opt because the A* approaches – limited by their simplicity – could solve more problems from the seq-opt track than the seq-sat track.

4.3 Implementations

We implement four different algorithms in a generic C++ framework, called Mosaic (see Appendix A), into which we originally intended to embed the search spaces of many C++ planners so that they could be compared on equal footing. Mosaic wraps the LAMA-2008 planner [RW10] that won the sixth International Planning Competition (IPC) in 2008. LAMA-2008 is based on the Fast Downward planner [Hel06].

We provide the implementation details of four algorithms: *Div* (Section 4.3.1), *ITA* (Section 4.3.2), and *Hybrid* (Section 4.3.3), and Random Walk Search (Section 4.3.4). Three algorithms extend A*, a common algorithm in the search planning literature (e.g., it is used as a core algorithm in LAMA [RW10], Fast Downward [Hel06], and FF [HN01]). A fourth algorithm implements a simplified variant of the Random Walk Search algorithm from Aarvand [XNM12]. All the algorithms employ only the neighborhood function of LAMA using the FF heuristic function and preferred operators; landmarks are not used (see Section 3.1

Table 4.1: Summarizing the domains used in this study.

Domain	Source	Problems	# Actions	# Predicates
`Depot`	IPC-2002	22	5	6
`Driverlog`	IPC-2002	20	6	6
`Rover`	IPC-2002	20	9	25
`Transport`	IPC-2011	20	3	5
`Cybersec`	IPC-2008			
		p01	96942	1710
		p02	96942	1710
		p03	96942	1710
		p04	96942	1710
		p05	96942	1710
		p06	69347	1143
		p07	69347	1143
		p08	1732	846
		p09	933	574
		p10	4564	535
		p11	4431	553
		p12	40284	992
		p13	69287	1105
		p14	69090	1105
		p15	97024	1718
		p16	96942	1710
		p17	96942	1710
		p18	96942	1710
		p19	96942	1710
		p20	96942	1710
		p21	69347	1143
		p22	69347	1143
		p23	1732	846
		p24	933	574
		p25	4564	535
		p26	4431	553
		p27	40284	992
		p28	69287	1105
		p29	69090	1105
		p30	97024	1718

for details about this planner). The neighborhood function of LAMA is used to generate the successors for the A* variants, which implement Weighted-A* (WA*) with a weight that begins at 10.0 and decreases by the factor 0.8 after a new solution is found.

Each algorithm is given 10 hours and up to 4 GB memory to produce up to 1000 plans for each problem. We chose 1000 plans with the hope that the algorithms would have trouble achieving such a high number, thus partially controlling for an artificial ceiling in the results. For this study, we did not verify that there were at least 1000 solutions for each problem, which is something we hope to do in future work. Each algorithm/problem pair is run on a single processor from one of 48 dual quad-core Xeon 5450, 16 GB machines. Algorithms terminate at 1000 plans or at memory or time limits.

4.3.1 Diversity-A* (*Div*)

Heuristic-based approaches modify the core heuristic of the algorithm to guide search toward diverse solutions. Coman and Muñoz-Avila [CMA11] guide search when using $D_{\text{stability}}$ (Equation 3.3) as the distance function, D, in Equation 3.1 to create a heuristic,

$$h_{\text{diversity}} = \text{RelativeDiversity}(\pi_{\text{relax}}, \Pi), \tag{4.4}$$

which measures the diversity of the current estimate of the best plan that can be achieved, π_{relax}, relative to the existing plan set Π found so far during the current search episode. π_{relax} is the final plan resulting from the relaxed plan heuristic, h^+ (see background, Section 3.2.1). The original heuristic h is combined with $h_{\text{diversity}}$ to create the heuristic used during search,

$$h_{\text{new}}(\pi, \Pi) = (1 - \alpha)h_{diversity} - \alpha h(\pi). \tag{4.5}$$

Note that h_{new} is maximized; the original heuristic is subtracted because the authors want to minimize the heuristic but maximize the diversity. The tuning parameter, α, balances exploitation of the original heuristic and exploration of more diverse plans. But nothing is done to control the scale between $h_{\text{diversity}}$ and h, so one value could still dominate the final value of h_{new} despite α. This is especially true as the number of solutions in Π increases.

Our implementation of the algorithm from Coman and Muñoz-Avila [CMA11] attempts to maintain as much similarity as possible with the original description while also keeping the comparison with our other WA* approaches fair. Similar to their planner, we leverage the $\text{D}_{stability}$ metric [Fox+06]. We also set $\alpha = 0.7$ as was done in their original studies

Table 4.2: Results on using $\pi_{current}$ (left) or π_{relax} (right) for *Div*.

$\pi_{current}$					π_{relax}			
n	Avg	Total	Unique		n	Avg	Total	Unique
9	206.2	1856	57	`Depot`	9	210.1	1891	40
12	423.3	5080	850	`Driverlog`	12	180.2	2163	77
20	263.2	5264	174	`Rover`	20	204.2	4083	63

[CMA11], although we found that the best value for α was highly dependent on the domain and problem; we discuss this as a limitation of our findings in Section 4.8. The heuristic used in our implementation of *Div* is:

$$h_{Div}(\pi, \Pi) = (2^{31}) - (1 - \alpha)\text{RelativeDiversity}(\pi_{Current}, \Pi) + \alpha h(\pi). \tag{4.6}$$

To maintain as much similarity as possible between the A* variants, we subtract the value of RelativeDiversity() from a large constant and then add back the original heuristic so that the algorithm minimizes h_{Div}.

We were concerned about the bias of using the relaxed plan to compute both h and $h_{\text{diversity}}$. So we explored using the current partial plan of the current search node, $\pi_{current}$, rather than the final estimated plan, π_{relax}, resulting from calculating the relaxed heuristic. Table 4.2 shows results comparing the use of $\pi_{current}$ (left) versus π_{relax} (right) for $\alpha = 0.70$. The columns in each sub-table indicate the total number of problems solved ('n'), the average number of plans produced per problem ('Avg'), the total number of plans produced for all the problems ('Total'), and the number of unique plans ('Unique') . Recall that unique solutions are those that are not permutations or subsets of other solutions (see Section 4.1). Using π_{relax} results in less unique solutions for `Depot`, `Driverlog`, and `Rover` problems. Yet it does not impact the coverage. This justifies using the current solution, $\pi_{current}$, in our implementation of *Div*.

4.3.2 Augmenting A* with a Tabu List (*ITA*)

State-based approaches directly control which states are visited during search. We designed an algorithm called Iterated Tabu A* (*ITA*) that augments the core A* algorithm

Algorithm 2 A*-TABU-SEARCH ($\mathcal{P}$, $\mathcal{T}$, maxSteps) returns a solution or failure

```
1: closed ← ∅
2: 𝒪 ← INSERT( MAKE-NODE( s_o ), 𝒪 )
3: stepsTaken ← 0
4: while stepsTaken ≤ maxSteps do
5:    if isEmpty( 𝒪 ) then
6:       return failure
7:    end if
8:    node ← 𝒪.removeNext()
9:    state ← node.getState()
10:   if S_g ⊆ state then
11:      return SOLUTION( node )
12:   end if
13:   if not inClosedList( state ) then
14:      closed ← closed ∪ state
15:      𝒪 ← 𝒪 ∪ EXPAND( node, 𝒫, 𝒯 )
16:   end if
17:   stepsTaken++
18: end while
```

with a Tabu list [Rob+12]. Algorithms 2 and 3 show the pseudocode for *ITA*; the only difference between Algorithm 2 and the original A* algorithm presented (cf. Algorithm 1) is in the addition of the Tabu list, $\mathcal{T}$ in line 15 and in the iteration in Algorithm 3. When the search finds a goal node, it calls EXTRACT-STATE-ACTION-PAIRS() on the current solution, which stores each $(state, operator)$ pair that is on the path to the solution into $\mathcal{T}$. *ITA* stores the entire state, which is a vector of variable assignments. On the next iteration, when the EXPAND() function encounters a $(state, operator)$ pair that is already in $\mathcal{T}$, *ITA* *adds to* the g-value of that node an arbitrarily large constant, $gConstant$. This ensures that those nodes are prioritized later in the A* open list and thus are very unlikely to be pulled off next. Our implementation of *ITA* uses $gConstant = 10^9$. There is no Tabu tenure so the Tabu list grows monotonically as more plans are found.

ITA was originally designed to find alternatives for the security application [Rob+11], where alternatives represent possible ways a system can be attacked and it is important to identify as many of these attacks as possible. We will show that *ITA* supports generating diverse solutions especially well for the security domain, and that it also can produce good alternatives for other benchmarks. However, it sometimes generates longer solutions that

Algorithm 3 ITERATED-TABU-A* ($\mathcal{P}$, numSolutions, maxSteps) returns a set of solutions or failure

```
1: solutions ← ∅
2: T ← ∅
3: while solutions.size() ≤ numSolutions do
4:     solution ← A*-SEARCH-TABU( P, T, maxSteps )
5:     T ← T ∪ EXTRACT-STATE-ACTION-PAIRS( solution )
6: end while
```

may be padded with spurious actions as a result of the state-based approach. We explore this tradeoff in the evaluation. *ITA* is not optimal because adding *gConstant* for nodes in the Tabu list can contradict the (assumed) admissibility of the heuristic. Further this version of A* does not reopen nodes with a better g-value from the closed list when they are rediscovered from the open list. But completeness and soundness remain unaffected, which is clear from the fact that the core A* algorithm remains unchanged.

4.3.3 Hybrid Diversity+Tabu (*Hybrid*)

There are drawbacks to both the heuristic-based and state-based approaches. Heuristic-based approaches can suffer because of the reliance on a single objective function; if action costs swamp the reward, then the search trajectory is dominated by the action cost. If two plans have wildly different action costs *and* different rewards, then it is not clear how to select one over the other. Further, the distance metric, D, of Equation 3.1 can dampen out the original heuristic, especially with a large difference in action costs. The state-based approach we propose avoids these scaling issues, but introduces others.

State-based approaches can suffer because of the lack of a guiding heuristic. Moreover, measuring diversity only in the state differences (or the actions applied) can generate solutions that cause too much surprise or disruption to a currently executing plan. *ITA* uses the entire state for the Tabu list rather than the portions relevant to the operator, which can lead to plans that have differences only in the variable binding of an operator (e.g., a package might be transported in a different truck that has the same fuel costs).

To help overcome the limitations of the heuristic and state approaches, we combine them into a single hybrid algorithm that performs both at the same time. The thinking behind

this hybrid approach is that the state-based Tabu list will help eliminate any dampening effect present in action costs because it focuses on the states visited during search. But the heuristic approach can also help the state-based approach by focusing the search on good solutions. We will see instead that these two approaches are not complementary and the hybrid produces the fewest number of solutions.

Our implementation of *Div+ITA* (*Hybrid*) enables the Tabu list of *ITA* while also using the diversity heuristic of *Div*. All other parameter settings remain the same as *Div* and *ITA*.

4.3.4 Random Walk Search

Random Walk Search (*RWS*) provides a natural baseline against which we can assess informed search because it selects actions without regard to their contribution to quality or diversity. Thus, it provides a fair way to assess where the bias of WA* and other algorithms is beneficial or harmful. The version of *RWS* we implement is inspired by the Arvand planner [XNM12]. *RWS* performs w walks where each walk performs p next descent probes of s steps. If an improving solution is not found after p probes then search restarts from the initial state. Our implementation of *RWS* performs $w = 1000$ walks where each walk performs $p = 7$ next descent probes of $s = 2000$ steps. The original algorithm in Arvand also incorporated probe lengthening enhancements [XNM12] that we do not use in our implementation. We expected *RWS* to produce more plans with greater diversity and lower quality than the other algorithms.

4.3.5 LPG-diffmax 2.0 (*LPGd*)

We complement the four algorithmic approaches with a full planning system. Srivastava et al. [Sri+07] explore how to generate diverse plans in a constraint-based planner called GraphPlan-CSP, and a local search planner, LPG-diffmax, that we abbreviate as *LPGd* in our study. In later work, these authors extended this work to planning with incomplete preference models [Ngu+12]. Both planners search for k plans of at least d distance apart. They use three normalized distance metrics based on actions, states, and causal links. For each distance metric, the authors embed it into the planner's core heuristic to encourage

diverse plans. At each decision point during search, the planner can insert a new action into the plan or remove an existing action from a partial plan it has constructed. Let $E(a)^i$ be the result of evaluating the insertion of an action and $E(a)^r$ be the result of evaluating the removal of an action in the plan. In the original version of LPG, the execution cost, temporal cost, and search cost are combined in a weighted evaluation function to determine which actions to insert or remove. In *LPGd*, an additional weighted term (i.e., the last term in the formulas below) is added to capture the contribution that the action provides from a reference plan π_0 that is selected at the start of each search iteration:

$$E(a)^i = \alpha_E \text{ExecutionCost}(a)^i + \alpha_T \text{TemporalCost}(a)^i + \alpha_S \text{SearchCost}(a)^i + \alpha_D |(\pi_0 - \pi) \cap \pi_R^i| \tag{4.7}$$

$$E(a)^r = \alpha_E \text{ExecutionCost}(a)^r + \alpha_T \text{TemporalCost}(a)^r + \alpha_S \text{SearchCost}(a)^r + \alpha_D |(\pi_0 - \pi - a) \cap \pi_R^r|. \tag{4.8}$$

Using these formulas, *LPGd* generates diverse plans. The authors then compare the resulting plan sets in terms of the three distance metrics, the average search cost to achieve the plans, and several preference functions. They focus on solving temporal metric and preference planning problems with large planning systems.

We obtained the code for the local search planner, *LPGd*, for inclusion in our study[9]; we could not obtain the constraint-based planner for this study. *LPGd* works by performing stochastic local search on planning graph subsets, which are partial plans that the authors call action graphs [GSS03; GS02]. At each branch in the search, the choice to add or remove an action is guided by a heuristic that rates each potential choice with respect to the current partial solution; the heuristic function for each potential action is the count of unsupported facts plus the count of actions which are mutex plus the diversity metric (Equation 3.5).

LPGd checks for and eliminates duplicates during its search, in contrast to the algorithmic approaches we examine. *LPGd* produces plan sets of size $k = \{2, 4, 8, .., 32\}$ that are at least d distance apart. To fit our experimental methodology, we modified the planner to produce 1000 solutions. However, *LPGd* would sometimes terminate (due to exhausting time or

[9]Personal communication with Tuan Nguyen and Ivan Serina

memory) and fail to produce intermediate results. The primary author of the planner[10] helped us modify the planner to use an increment policy for k that starts at 20 and increments by 20 up to 1000 after each plan set is found. We use $d = 0.5$ because this appeared to be a reasonable value given previous results [Ngu+12]. However, we acknowledge that it may be unfair to compare *LPGd* in this experimental methodology, and in Section 4.8 we are careful to limit our findings concerning our use of *LPGd*.

4.4 Results: Parsimony, Uniqueness, and Overlap

We now compare the approaches for generating plan sets using the metrics we outlined at the outset of this chapter. Recall that we focus our analysis on the IPC2002 domains `Depot`, `Driverlog`, and `Rover`, on the IPC2008 `Cybersec` domain, and on the IPC2011 `Transport` domain. Also recall that our analysis evaluates quality and efficiency by using the coverage, search cost, plan length, and plan cost. In the 2002 domains (i.e., `Depot`, `Driverlog`, and `Rover`) we use plan length for plan quality and in the other two we use plan cost; this is a result of some domains being older and not having plan cost. For diversity, we evaluate the plan sets using the parsimony ratio, uniqueness and overlap, and the two distance metrics $Diversity_{\text{stability}}$ (D_s, Equation 3.3) and $Diversity_{\text{norm}}$ (D_n, Equation 3.5).

4.4.1 Parsimony

Parsimony is a central concept to understanding our results. Although we discovered it near the end of our exploration, we present it first here because it informs (and colors) much of what we find. Plans produced by nearly every approach consistently contain action cycles. *LPGd* was the most verbose and often has (10-30) extra actions in a given plan. Figure 4.2 shows one example, where a plan from *ITA*, *RWS*, and *LPGd* are shown, respectively. The plan from *LPGd* is annotated with asterisks ('*') to show a 2-way action sequence; note the

[10] Personal communication, Ivan Serina

repeated inclusion of `lift` and `drop` actions for `crate0`. We have strong reason to believe that even higher n-way action sequences occur within many plans of the approaches we studied.

To understand how much longer plans are, we calculate the parsimony ratio where possible. Table 4.3 summarizes the distributions (mean, standard deviation, median and minimum ratios) for the s-ratio on the solutions for which we could find a minimal plan. We apply the Tukey HSD statistical test (α=0.05, df=4) to determine differences between the means of the ratio distributions for the algorithms. Tukey HSD performs the standard t-test between all pairs of mean ratios while simultaneously controlling the experimentwise error that can result from performing multiple pair-wise inferences; it conservatively shows a significant difference between means. We summarize the groupings that are **not** significantly different, so algorithms within a group perform similarly. In every domain, the A* algorithms are grouped as similar (i.e., {*Div*, *Hybrid*, *ITA*}). *LPGd* and *RWS* are distinct from each other in every domain except `Transport`.

We can see that *LPGd* and *RWS* produce the lowest average and minimum ratios, indicating that plans often contain extraneous actions. Both of these algorithms also have the highest variance, indicating that a wider range of parsimony ratio values is seen in the solutions. In contrast, the algorithmic approaches produce minimal plans more often (higher means) with more reliability (lower variance). The algorithmic approaches also show a better minimum performance.

We will show in the following sections that the approaches with the lowest parsimony achieve higher diversity and uniqueness. These action cycles ensure the uniqueness of the plans under our analysis and they artificially inflate the diversity of the plan set. When paired with the overlap and number of unique solutions, the parsimony ratio provides strong evidence that *LPGd* and *RWS* obtain such high uniqueness values by including n-way action cycles in the plan that achieve uniqueness and diversity at the cost of plan parsimony. Overlap is probably artificially low as well because of these action cycles. This observation of parsimony will play an important role in our design decisions for *MQA* in Section 5.3.2.

```
(lift hoist0 crate1 pallet0 depot0)
(load hoist0 crate1 truck1 depot0)
(drive truck1 depot0 distributor0)
(lift hoist1 crate0 pallet1 distributor0)
(load hoist1 crate0 truck1 distributor0)
(unload hoist1 crate1 truck1 distributor0)
(drop hoist1 crate1 pallet1 distributor0)
(drive truck1 distributor0 distributor1)
(unload hoist2 crate0 truck1 distributor1)
(drop hoist2 crate0 pallet2 distributor1)
```

```
(lift hoist0 crate1 pallet0 depot0)
(drive truck0 distributor1 distributor0)
(lift hoist1 crate0 pallet1 distributor0)
(drop hoist0 crate1 pallet0 depot0)
(lift hoist0 crate1 pallet0 depot0)
(load hoist0 crate1 truck1 depot0)
(drive truck1 depot0 distributor0)
(load hoist1 crate0 truck0 distributor0)
(unload hoist1 crate1 truck1 distributor0)
(drive truck0 distributor0 distributor1)
(drop hoist1 crate1 pallet1 distributor0)
(unload hoist2 crate0 truck0 distributor1)
(drop hoist2 crate0 pallet2 distributor1)
```

```
 (drive truck0 distributor1 distributor0)
 (lift hoist0 crate1 pallet0 depot0)
 (lift hoist1 crate0 pallet1 distributor0)
 (drive truck0 distributor0 depot0)
 (load hoist0 crate1 truck0 depot0)
 (drive truck0 depot0 distributor0)
 (drive truck0 distributor0 distributor1)
 (unload hoist2 crate1 truck0 distributor1)
 (drop hoist2 crate1 pallet2 distributor1)
 (drive truck0 distributor1 distributor0)
 (load hoist1 crate0 truck0 distributor0)
 (drive truck0 distributor0 depot0)
 (unload hoist0 crate0 truck0 depot0)
 (load hoist0 crate0 truck1 depot0)
 (drive truck1 depot0 distributor0)
 (drive truck1 distributor0 distributor1)
 (unload hoist2 crate0 truck1 distributor1)
 (drop hoist2 crate0 crate1 distributor1)
*(lift hoist2 crate0 crate1 distributor1)
*(drop hoist2 crate0 crate1 distributor1)
*(lift hoist2 crate0 crate1 distributor1)
*(drop hoist2 crate0 crate1 distributor1)
*(lift hoist2 crate0 crate1 distributor1)
*(load hoist2 crate0 truck1 distributor1)
 (lift hoist2 crate1 pallet2 distributor1)
 (load hoist2 crate1 truck1 distributor1)
 (unload hoist2 crate0 truck1 distributor1)
 (drop hoist2 crate0 pallet2 distributor1)
 (drive truck1 distributor1 depot0)
 (unload hoist0 crate1 truck1 depot0)
 (load hoist0 crate1 truck0 depot0)
 (drive truck0 depot0 distributor0)
 (unload hoist1 crate1 truck0 distributor0)
 (drop hoist1 crate1 pallet1 distributor0)
```

Figure 4.2: The first solution by *ITA*(lop left), *RWS* (right), and *LPGd* (bottom left) for `Depot`, problem 1. The asterisks indicate one place where the solution by *LPGd* contained an extraneous two-way action cycle.

4.4.2 Uniqueness and Overlap

Table 4.4 shows solution counts for each algorithm by domain. The p column displays how many problems the algorithm solved for a domain followed by *Avg*, the average number of plans generated in those domains. *Total* is how many plans each algorithm generated and

Table 4.3: Distributions of the parsimony ratio, s, for each domain and algorithm. Search could not find the minimal plan π_k for `Cybersec`, which is indicated by a dash ('-').

	$\bar{x}$	SD	Med	Min
RWS				
Depot	0.74	0.106	0.75	0.34
DriverLog	0.64	0.121	0.63	0.25
Rover	0.82	0.114	0.81	0.44
Cybersec	-	-	-	-
Transport	0.74	0.096	0.74	0.37
Div				
Depot	0.93	0.089	1.00	0.77
DriverLog	0.98	0.046	1.00	0.81
Rover	1.00	0.002	1.00	0.90
Cybersec	-	-	-	-
Transport	0.98	0.043	1.00	0.82
ITA				
Depot	1.00	0.018	1.00	0.72
DriverLog	0.99	0.040	1.00	0.61
Rover	1.00	0.011	1.00	0.83
Cybersec	-	-	-	-
Transport	0.95	0.070	1.00	0.50
Hybrid				
Depot	0.93	0.095	1.00	0.75
DriverLog	0.97	0.045	1.00	0.88
Rover	1.00	0.000	1.00	1.00
Cybersec	-	-	-	-
Transport	0.95	0.055	0.98	0.83
LPGd				
Depot	0.53	0.233	0.44	0.18
DriverLog	0.82	0.142	0.86	0.20
Rover	0.95	0.054	0.96	0.72
Cybersec	-	-	-	-
Transport	0.73	0.165	0.76	0.16

Unique is the number of unique plans generated; *Ratio* is $Unique/Total$. The best overall approaches are noted in bold. An asterisk indicates the best algorithmic (A* variant or *RWS*) approach. The last five columns compare the overlap between the approaches.

Table 4.4: Solution counts and overlap for the domains on *RWS*, *Div*, *ITA*, *Hybrid*, and *LPGd*. The n column indicates the number of problems that were solved at least once for each domain by an algorithm. The right set of columns show the number of *plans* that overlap between pairs of approaches.

	n	Avg	Total	Unique	Ratio	*RWS*	*Div*	*ITA*	*Hybrid*	*LPGd*
				RWS						
Depot	12	376.4	*4517	*3181	0.704		0	1	1	262
DriverLog	17	776.5	***13201**	*11011	0.834		0	0	0	2
Rover	14	706.6	***9893**	*6232	0.630		0	1	1	4
Cybersec-strips	21	5.6	118	118	1.000		0	0	0	0
Transport	20	734.1	*14682	*14549	0.991		0	0	0	19
				Div						
Depot	8	231.0	1848	54	0.029	0		11	18	57
DriverLog	12	423.6	5083	855	0.168	0		9	26	64
Rover	20	275.3	5506	175	0.032	0		29	37	2
Cybersec	28	143.7	4023	563	0.140	0		94	340	5
Transport	20	576.0	11519	4356	0.378	0		10	481	14
				ITA						
Depot	8	282.9	2263	246	0.109	1	11		13	140
DriverLog	13	205.2	2667	375	0.141	0	9		14	98
Rover	20	226.7	4533	314	0.069	1	29		33	5
Cybersec-strips	28	261.8	7331	***1562**	0.213	0	94		103	4
Transport	20	329.8	6595	5467	0.829	0	10		9	17
				Hybrid						
Depot	8	262.5	2100	100	0.048	1	18	13		130
DriverLog	12	210.1	2521	321	0.127	0	26	14		127
Rover	20	228.1	4562	382	0.084	1	37	33		3
Cybersec	28	220.6	6177	1342	0.217	0	340	103		5
Transport	20	332.4	6649	5574	0.838	0	481	9		23
				LPGd						
Depot	22	819.9	**18038**	**17991**	0.997	262	57	140	130	
DriverLog	20	582.9	11657	**11650**	0.999	2	64	98	127	
Rover	20	432.2	8644	**8644**	1.000	4	2	5	3	
Cybersec-strips	24	48.0	1152	1152	1.000	0	5	4	5	
Transport	20	974.9	**19498**	**19491**	1.000	19	14	17	23	

LPGd solves more problems in `Depot` and `Driverlog` than the algorithmic approaches. It also generates the largest total of solutions for `Depot` and `Transport` and generates the

most unique plans in every domain except `Cybersec`. Finally, it produces the highest unique ratio of the algorithms. There are several possible explanations for the performance gap between the algorithmic approaches and *LPGd*. First, *LPGd* skips reporting of duplicate solutions during its search. Skipping duplicates during search has the benefit of making the final uniqueness evaluation better, but this comes at an increased search cost for novel plans.

When comparing just the algorithmic approaches (i.e, the entries with an asterisk) to each other, *RWS* finds more plans and more unique plans for the IPC-2002 and `Transport` benchmarks than the A* approaches. This may be a consequence of its simplicity (very low memory overhead and lack of open/closed lists) so it can process more potential plans. As mentioned in Section 4.8, the algorithmic approaches fail to solve as many problems as a state-of-the-art planner; this is evident by examining the n column between the algorithms and *LPGd*. This is likely due to reverting to only WA* without other search enhancements such as lazy initialization or multiple queues. For `Cybersec`, we note that the plan lengths are quite long (40–60 steps) and it is likely that *RWS* has trouble finding these longer plans.

Many plans are duplicates, and the unique-to-total ratio varies a great deal between the algorithms. *Div* has the lowest ratio of the WA* algorithms except in `Driverlog` while *RWS* has the highest ratio of all algorithms. In `Cybersec` and `Transport`, both *ITA* and *Hybrid* have similar ratios. *ITA* finds the most (unique) plans in `Cybersec`, which is expected since the algorithm was originally designed to produce alternatives for a security domain very similar to `Cybersec`. Except in `Driverlog`, *ITA* produces more unique plans than *Hybrid*, which produces more unique plans than *Div*.

The overlap columns show that each algorithm generates unique plans not found by the others as noted by the generally low intersection. For the algorithmic approaches, the intersection is higher among the WA* algorithms compared to *RWS*. The greater overlap of *Hybrid* with *Div* suggests that adding the diversity heuristic to *ITA* provides more benefit than adding a Tabu list to *Div*. *LPGd* has the greatest overlap with the other approaches for `Depot`, `Driverlog`, and `Transport` – this is most likely the result of *LPGd* having generated more solutions.

4.4.3 Diversity

Table 4.5 compares the algorithms using Diversity(Π) from Equation 3.3, while Table 4.6 compares the algorithms using Diversity(Π) from Equation 3.5. These tables have two types of entries: 1) when an approach is compared to itself and 2) when an approach is compared to a different algorithm. When an algorithm is compared with itself, the number of problems for which 10 unique solutions were found is presented. For example, in Table 4.5 the top-left entry for *Div*—*Div* shows that *Div* only had 4 problems where it produced at least 10 unique solutions; a quick glance at Table 4.4 verifies that *Div* produced 54 unique solutions over 8 problems.

When two algorithms are compared, we take their union. Each entry shows how often the row approach produced equal or higher diversity than the column approach ('$\geq$'), the size of the union ('$\cup$'), and their average magnitude of difference $|\bar{\delta}|$. Since we take the union of problems with at least 10 unique solutions, any approach that does not solve a problem gets a score of 0 for its diversity. For example, the first row of the second column (again in Table 4.5) shows the *Div*—*Hybrid* comparison. It states that 7 problems were solved by both algorithms, and *Div* produced higher diversity for 4 of the 7 problems. Further, the magnitude of difference for all 7 problems was 12.7. Continuing the example, we see a similar trend when *Div* is compared to *ITA* but that both *LPGd* and *RWS* produce more unique solutions than *Div*. The magnitude for *LPGd* is also higher, but this is again more a consequence of *LPGd* solving more problems.

RWS solution sets have higher diversity than the other algorithmic approaches except in `Cybersec`. This finding seems to contradict Nguyen et al. [Ngu+12], who state that their randomized approach produces *less* diverse plans than *LPGd* using the plan distance metrics. Figure 4.2 shows an example of the extra action cycles. Without comparing the actual solutions from the original experiments, it is difficult to determine the impact of the extra actions in the results from [Ngu+12]. But it is likely that their random approach fared worse because it included fewer extraneous actions than the standard *LPGd* approach.

Table 4.5: Average stability diversity (D_s) comparison for the first 10 unique plans for all algorithms. Each entry shows how often the column algorithm's average diversity for those ten solutions was better or equal to the row algorithm's average diversity ('$\geq$' columns), the union of problems with at least 10 solutions in either algorithm ('$\cup$' columns), and the average magnitude of difference ('$|\bar{\delta}|$' columns).

	Div			*Hybrid*			*ITA*			*LPGd*			*RWS*		
	$\geq$	$\cup$	$\lvert\bar{\delta}\rvert$	$\geq$	$\cup$	$\lvert\bar{\delta}\rvert$	$\geq$	$\cup$	$\lvert\bar{\delta}\rvert$	$\geq$	$\cup$	$\lvert\bar{\delta}\rvert$	$\geq$	$\cup$	$\lvert\bar{\delta}\rvert$
Depot															
Div		4		4	7	12.7	4	8	11.8	0	22	22.3	1	11	8.8
Hybrid	3	7	12.7		7		0	8	1.4	0	22	51.2	0	11	35.4
ITA	4	8	11.8	8	8	1.4		8		0	22	51.7	1	12	34.0
LPGd	22	22	22.3	22	22	51.2	22	22	51.7		22		20	22	23.0
RWS	10	11	8.8	11	11	35.4	11	12	34.0	2	22	23.0		11	
Driverlog															
Div		6		6	12	5.6	6	12	5.4	0	20	10.1	1	17	6.3
Hybrid	7	12	5.6		12		6	13	3.9	0	20	24.0	1	18	48.1
ITA	7	12	5.4	9	13	3.9		12		1	20	45.1	2	19	41.3
LPGd	20	20	10.1	20	20	24.0	19	20	45.1		20		10	20	28.8
RWS	16	17	6.3	17	18	48.1	17	19	41.3	10	20	28.8		17	
Rover															
Div		10		10	19	9.1	10	19	10.0	1	20	4.6	4	13	5.3
Hybrid	10	19	9.1		19		10	19	2.9	0	20	19.5	8	20	14.3
ITA	10	19	10.0	10	19	2.9		19		0	20	19.2	8	20	14.4
LPGd	19	20	4.6	20	20	19.5	20	20	19.2		20		16	20	3.3
RWS	10	13	5.3	12	20	14.3	12	20	14.4	4	20	3.3		12	
Cybersec															
Div		26		21	28	10.4	18	28	10.0	8	28	16.3	14	26	13.3
Hybrid	8	28	10.4		28		12	28	6.2	5	29	21.1	14	28	11.1
ITA	10	28	10.0	16	28	6.2		28		7	29	16.5	17	28	14.4
LPGd	20	28	16.3	24	29	21.1	22	29	16.5		24		18	25	16.7
RWS	13	26	13.3	14	28	11.1	11	28	14.4	7	25	16.7		19	
Transport															
Div		17		9	20	9.2	9	20	9.0	0	20	33.7	0	20	36.8
Hybrid	12	20	9.2		20		13	20	1.0	0	20	30.6	0	20	33.1
ITA	13	20	9.0	10	20	1.0		20		0	20	30.7	0	20	33.2
LPGd	20	20	33.7	20	20	30.6	20	20	30.7		20		9	20	11.1
RWS	20	20	36.8	20	20	33.1	20	20	33.2	11	20	11.1		20	

Table 4.6: Average normalized diversity (D_n) comparison for the first 10 unique plans for all algorithms. Each entry shows how often the column algorithm's average diversity for those ten solutions was better or equal to the row algorithm's average diversity ('$\geq$' columns), the union of problems with at least 10 solutions in either algorithm ('$\cup$' columns), and the average magnitude of difference ('$|\bar{\delta}|$' columns).

	Div			*Hybrid*			*ITA*			*LPGd*			*RWS*		
	$\geq$	$\cup$	$\lvert\bar{\delta}\rvert$	$\geq$	$\cup$	$\lvert\bar{\delta}\rvert$	$\geq$	$\cup$	$\lvert\bar{\delta}\rvert$	$\geq$	$\cup$	$\lvert\bar{\delta}\rvert$	$\geq$	$\cup$	$\lvert\bar{\delta}\rvert$
Depot															
Div		3		2	3	0.1	2	7	0.1	1	22	0.2	2	9	0.1
Hybrid	3	3	0.1		3		3	7	0.1	1	22	0.1	3	9	0.0
ITA	7	7	0.1	6	7	0.1		7		2	22	0.2	3	10	0.1
LPGd	22	22	0.2	22	22	0.1	22	22	0.2		22		21	22	0.1
RWS	9	9	0.1	8	9	0.0	9	10	0.1	3	22	0.1		9	
Driverlog															
Div		6		4	6	0.3	5	12	0.2	3	20	0.3	3	17	0.3
Hybrid	4	6	0.3		6		5	12	0.1	4	20	0.2	4	17	0.2
ITA	10	12	0.2	10	12	0.1		12		7	20	0.1	9	19	0.1
LPGd	19	20	0.3	18	20	0.2	17	20	0.1		20		18	20	0.1
RWS	14	17	0.3	13	17	0.2	13	19	0.1	9	20	0.1		17	
Rover															
Div		5		5	5	0.1	2	19	0.1	1	20	0.2	2	12	0.1
Hybrid	2	5	0.1		4		2	19	0.1	1	20	0.2	0	12	0.1
ITA	17	19	0.1	18	19	0.1		19		3	20	0.2	11	20	0.2
LPGd	20	20	0.2	20	20	0.2	20	20	0.2		20		18	20	0.1
RWS	12	12	0.1	12	12	0.1	10	20	0.2	5	20	0.1		12	
Cybersec															
Div		2		1	11	0.1	1	28	0.1	0	24	0.2	1	19	0.1
Hybrid	10	11	0.1		10		5	28	0.2	4	26	0.2	3	21	0.1
ITA	28	28	0.1	23	28	0.2		28		8	29	0.2	17	28	0.2
LPGd	24	24	0.2	24	26	0.2	23	29	0.2		24		22	25	0.1
RWS	19	19	0.1	19	21	0.1	13	28	0.2	7	25	0.1		19	
Transport															
Div		6		4	6	0.1	5	20	0.1	4	20	0.1	4	20	0.1
Hybrid	5	6	0.1		6		5	20	0.1	4	20	0.1	4	20	0.1
ITA	18	20	0.1	16	20	0.1		20		13	20	0.1	11	20	0.1
LPGd	18	20	0.1	18	20	0.1	17	20	0.1		20		18	20	0.0
RWS	19	20	0.1	19	20	0.1	12	20	0.1	14	20	0.0		20	

None of the WA* algorithms dominate each other in producing more diversity. The relatively poor performance of *Div* may be due to the fact that the diversity values detract from the original heuristic. The diversity values seen during search will be different than those we can observe in a post-hoc analysis of complete solutions; recall that we use the partial solution $\pi_{current}$ during search. When we examine the pairwise diversity values for Π given by Equation 3.3, which is an average, we note that the range of values seen by *RWS* appears to be much larger than the other three algorithms. Further, there is an extreme skew of values toward the low end, which can result from a planner producing duplicated plans. This suggests that 1) algorithms that produce numerous duplicate plans actually dilute the value of using diversity to guide search, and 2) a "solution Tabu" mechanism may help eliminate the dilution for search algorithms employing diversity. Part of the success of *LPGd* in producing greater diversity is no doubt due to it checking for and skipping over duplicate solutions.

4.5 Results: Quality

We examine the quality of the plans as measured by VAL [HL03], which is the program used to validate plans in the IPCs. For all the domains, the quality metric is minimized. `Transport` minimizes the total cost of moving packages, where the cost is a sum of the road distances plus 1 for each unload/load action. `Cybersec` minimizes the total cost of actions, and the other three domains minimize plan length. Figure 4.3 shows the distributions of quality values of each algorithm for the first 10 unique plans; these values are normalized by the maximum value in each problem for all approaches.

We apply the Tukey HSD statistical test (α=0.05, df=4) to determine significant differences between the problem-normalized quality of the approaches, and report the resulting groupings. For `Cybersec`, there was no significant difference between the quality of plans produced and thus a pairwise comparison is unjustified. For `Depot`, `Driverlog`, and `Rover` the same two trends occurred: 1) all three A* algorithms produced quality that was distinct from (and lower than) *LPGd* and *RWS*, and 2) *Div* performed the same as *Hybrid* and *ITA* but

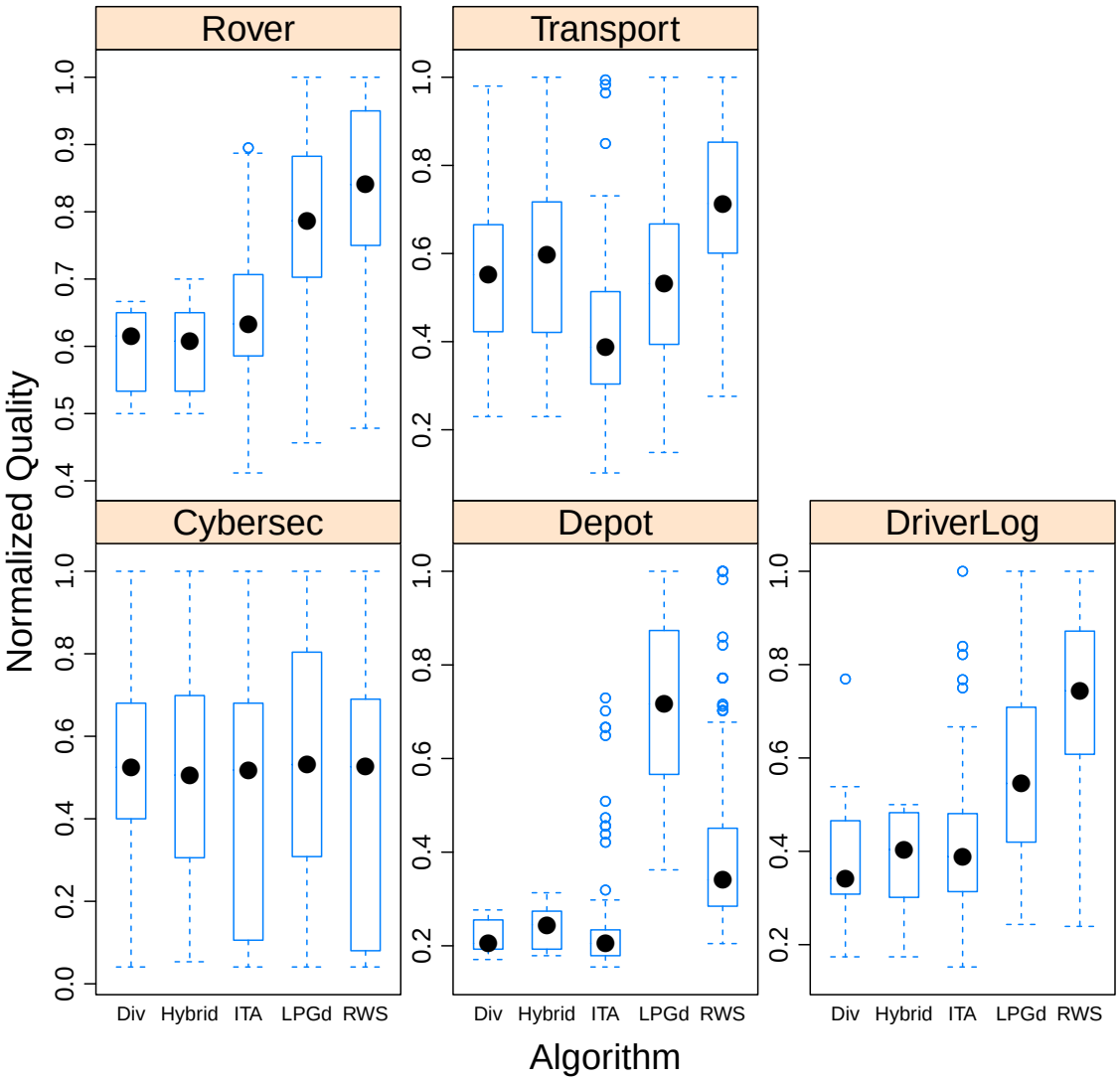

Figure 4.3: Comparing normalized quality of the first 10 unique plans.

Hybrid and *ITA* were distinct from each other. *LPGd* produced significantly better quality solutions than *RWS* except for `Depot`; it is clear that *LPGd* fares worse because it produces much less parsimonious plans (cf. Table 4.3). In the `Transport` domain, *LPGd* produced about the same quality solutions as *Div* and *Hybrid*. Thus, while the three WA* algorithms generally produce much better quality plans than *LPGd* and *RWS*, that is not always the case. The high degree of similarity between *Hybrid* and the other algorithms suggests that *Hybrid* does not improve quality over the other approaches.

4.6 Results: Search Cost

We examine search cost for the algorithmic approaches. *LPGd* is not included because it does not keep track of the cost *per solution*; we intend to add this to the code, rerun *LPGd*, and incorporate the search cost estimates for *LPGd* in future work. Search cost is plotted in Figure 4.4 (bottom, log scale); as before, `Driverlog` was similar to `Depot` and the Tukey HSD groupings are shown in red (gray). For `Cybersec`, the Tukey HSD (α=0.05, df=3) groupings were {*Div*, *Hybrid*}, {*Hybrid*, *ITA*}, and {*RWS*}. All other domains were grouped {*Div*, *Hybrid*, *ITA*} and {*RWS*}. *RWS* is always significantly different and produces plans after a large number of evaluations, which is an expected finding. The three WA* variants tend to behave similarly, which suggests that the α weight in the diversity heuristic does not lower the usefulness of the original heuristic so much that search moves into non-productive parts of the search tree. In general the hybrid algorithm did not show a significant performance difference from *Div* and *ITA*. Even in `Cybersec`, the difference between *Div* and *ITA* while both being similar to *Hybrid* shows that the hybrid algorithm's performance is comparable to both *Div* and *ITA* algorithms despite them not being similar.

4.7 Results: Security Domains

In terms of the security domains we examined, Table 4.4 shows that *ITA* produces the most solutions for the `Cybersec` domain. We also evaluate how well plans can be found for the motivating security application. *ITA* can find unique solutions (alternative attack paths) for the small PAG domain generated from Figure 2.1 as well as the `Cybersec` benchmark domain.

Figure 4.5 (top) shows a partial PDDL description for the security domain motivated in Chapter 2; the full PDDL description is given in Figure B.1. In codifying the PAG, we followed a similar approach to the prior work by translating the PAG into PDDL [Rob+11]. Figure 4.5 shows portions of the domain and problem description of the leftmost subtree given in Figure 2.1. The full PDDL description is provided in Figure B.1 and contains 15

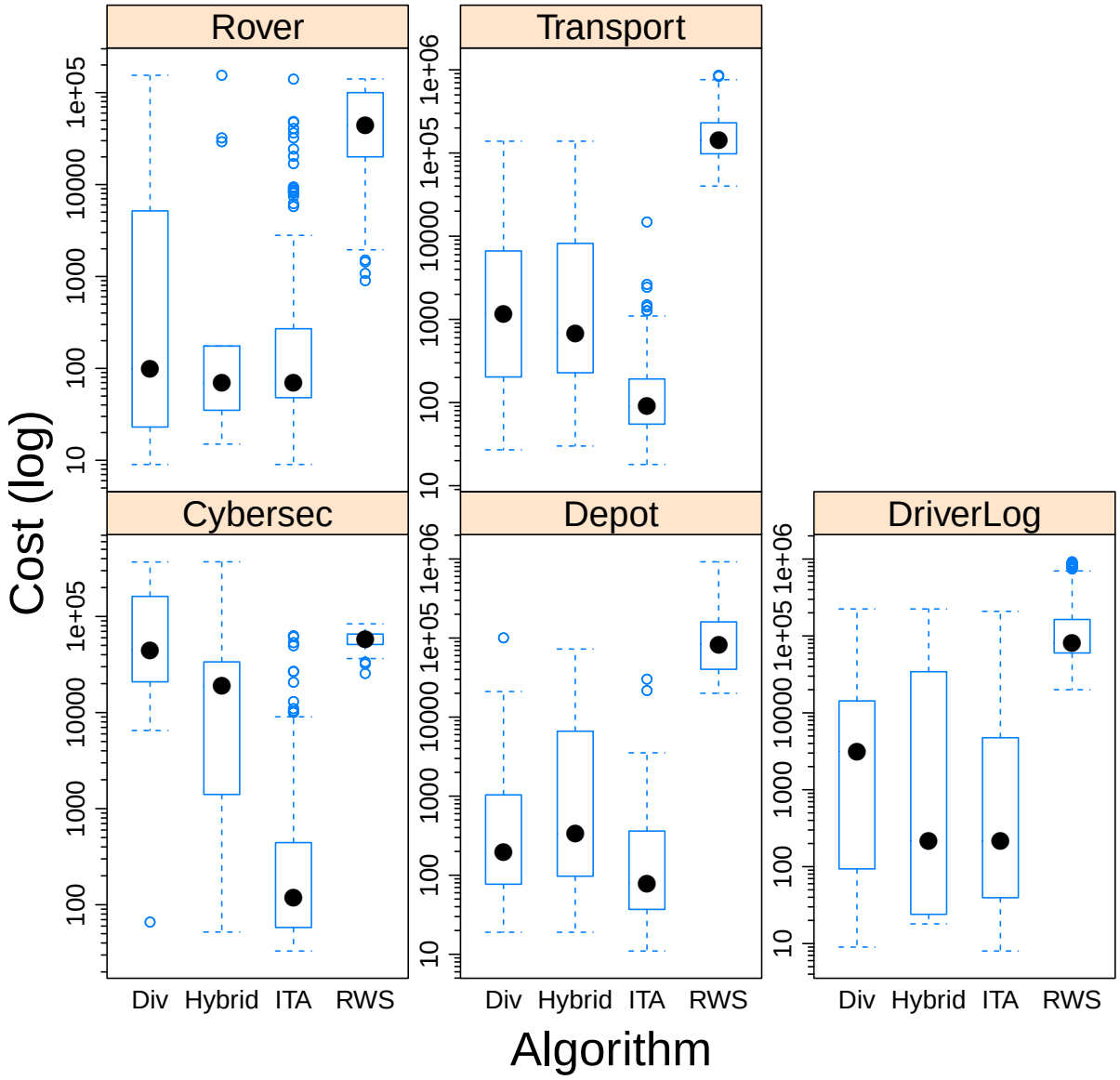

Figure 4.4: Comparing search cost (log scale) of the first 10 unique plans.

actions, 5 predicates, 18 initial objects, and 12 initial predicates; these sizes are comparable to the benchmark domains except `Cybersec` (c.f. Table 4.1). Plans in this domain highlight potential paths that can be exploited, which allows other portions of the agent to prune and personalize the PAG and remove exploits that cannot happen for a given system/user.

Figure 4.5 (bottom) shows the three solutions found by *ITA* in a single planning episode, which correspond to the three subtrees of the PAG. It found these solutions with 53, 46,

```
(define (domain attack-graph)
    (:requirements :strips :equality
       :disjunctive-preconditions :typing )
    (:types    Object   Action   ExploitState   software )
    (:predicates   (action-observed ?Action - Action)
      (action-taken ?Action - Action)
      (exploit-occurred  ?Exploit - ExploitState)
      (software-installed  ?Software - software) )
    (:action AttackAction_FlashFileCompromised_5
      :parameters ( ?Action5 - Action )
      :precondition (and (action-observed ?Action5 )
        (= ?Action5 AttackAction_FlashFileCompromised_5 ))
      :effect
        (and
        (action-taken AttackAction_FlashFileCompromised_5))))

(define (problem attack-graph-problem1)
    (:domain attack-graph)
    (:objects
      Exploit_DenialOfService_1 - ExploitState
      AttackAction_FlashFileCompromised_5 - Action
      UserAction_UserUsingSocialMedia_7 - Action
      ObeservedState_CVE_2010_0187_Exploited_2 - Action
      UserAction_UserOpensFlashFile_6 - Action
    )
    (:init
      (action-observed AttackAction_FlashFileCompromised_5 )
      (action-observed UserAction_UserUsingSocialMedia_7 )
      (software-installed Adobe_Flash_6_0_88_0)
    )
    (:goal
      (and (exploit-occurred Exploit_DenialOfService_1) )))

PLAN1
UserAction_UserUsingSocialMedia_7
UserAction_UserOpensFlashFile_6
ObeservedState_CVE_2010_0187_Exploited_2
Exploit_DenialOfService_1

PLAN2
UserAction_UserBrowsingInternetContent_13
AttackAction_PDFCompromised_20
UserAction_UserLoadsPDFDocument_21
ObeservedState_CVE_2010_4091_OS_Exploited_17
Exploit_DenialOfService_1

PLAN3
UserAction_UserBrowsingInternetContent_13
AttackAction_JavaAppWithLongVMArgument_11
UserAction_UserStartsJavaWebstartApplication_12
ObeservedState_CVE_2008_3111_SunJavaMultiple_Exploited_8
Exploit_DenialOfService_1
```

Figure 4.5: Partial PDDL domain and problem descriptions from the CVE-2010-0187 subtree of the DoS exploit from Figure 2.1 followed by the three *ITA* solutions. The full PDDL description is given in Figure B.1.

and 25 node expansions. The plan lengths for these solutions increase from 4 to 5, so *ITA* overcomes a key problem identified by both Boddy et al. [Bod+] as well as Ghosh and Ghosh [GG12], that the planner always finds the shortest path. For this problem, *Div* found a single

path but was unable to find new plans beyond that. We suspect that either the weighted heuristic or the diversity measure is not well suited to small plans and domains such as those of the security application. Plans in this domain are typically 5-10 actions total, while the focus in the benchmarks is on scalability of planners to plan lengths of 20 or more actions.

4.8 Limitations

In earlier work on evaluating planner performance [RH09b], we found that the use of full planning systems obscured the planner component most linked with search performance. So in this study we sought to use simple algorithmic approaches to make the comparison more understandable. However, this decision led to less potential data points for making our comparisons – simple algorithms lack the refinements that increase the coverage of planners. We dealt with this limitation by comparing the approaches on problems they solved in common, by using unique solutions, and by limiting the solutions compared.

Another limitation of our findings is our use of the LAMA-2008 planner as the base planner for Mosaic, because by now a newer version of LAMA is available that won the subsequent IPC-2011. Although Mosaic was designed to make adding planner (or planner versions) easy, it turns out to be much more difficult to port planners into Mosaic than we anticipated for a number of unforeseen reasons; we discuss this "hidden" challenge in Section 6.4. It would be a significant effort to report results for LAMA-2011 here. Since we are not using a state-of-the-art planning system, the results from the thesis cannot be compared to results from the latest IPC planning systems in terms of the quality and efficiency axes of our study. However, our use of an established heuristic, h_{FF}, and the fact that all our algorithms are implemented within the same framework should generalize, since we seek answers concerning the relative differences between these algorithms while using the same underlying planner foundation.

With regard to *LPGd*, we used the default parameters listed in the most recent published results [Ngu+12]. However, different parameters might yield better results; we did not perform an exhaustive parameter search to determine the best parameters for the problems

we studied. Nguyen et al. presented two additional mechanisms for generating diverse plans using causal links and action states. These approaches may be able to drive search toward diversity in a more consistent fashion that eliminates some of the issues we note in our results. We note that *LPGd* was not run to produce anywhere near 1000 solutions for those original studies. Finally, *LPGd* was designed to work on temporal and preference domains, and it may be completely unfair to use *LPGd* in our experimental methodology.

With regard to the *Div* and *Hybrid* algorithms that use the weighted diversity values within the planner heuristic, we observed two limitations. The first relates to the chosen value for α for *Div*; recall that we set $\alpha = 0.70$ to replicate the original studies. However, in follow up experiments, we vary the α value for *Div*. Table 4.7 (left) shows the results for the `Rover`, `Depot`, and `Driverlog` problems, which are the same domains used in the original studies [CMA11]. The table includes results for $\pi_{current}$ and π_{relax} as discussed in Section 4.3.1. The columns show the alpha value ('α'), the number of problems solved with at least one solution ('n'), the average number of solutions produced per problem solved ('Avg'), the total number of solutions produced ('Total'), the number of unique solutions ('Unique'), and the ratio of unique/total ('Ratio'). In each column, the highest value for the particular domain is in bold face font. The Unique/Total Ratio shows that the best α value for *Div* depends highly on the domain. In some cases, more problems ('n' column) are solved by certain values of α. It is evident that *for our implementation of this algorithm* $\alpha = 0.70$ is rarely the best value. It remains as future work to explain why our results do not match with the original implementation. Even if we were to include the very best alpha value for *Div* our conclusions remain the same. So this limitation, while important to admit, does not alter the substance of our findings with respect to the tradeoffs in generating plan sets.

It is interesting to note that Table 4.7 shows that neither pure heuristic search (i.e., $\alpha = 1.00$) nor pure diversity search ($\alpha = 0.00$) are the best approaches. Although it is clear that varying the heuristic can result in more unique solutions for `Rover` and `Depot`, the best unique ratios result from a weighted combination of the original heuristic with the diversity heuristic.

Table 4.7: Results on varying α using $\pi_{current}$ (left) and π_{relax} (right) for *Div*.

	$\pi_{current}$					π_{relax}				
α	n	Avg	Total	Unique	Ratio	n	Avg	Total	Unique	Ratio
				`Rover`						
0.00	8	771.6	6173	146	0.024	7	572.0	4004	24	0.006
0.10	7	810.6	5674	94	0.017	7	581.0	4067	51	0.013
0.20	**20**	381.7	7634	879	0.115	**20**	204.7	4093	57	0.014
0.30	**20**	**389.6**	**7792**	803	0.103	**20**	204.4	4088	70	0.017
0.40	**20**	372.8	7456	**1147**	**0.154**	**20**	204.2	4085	61	0.015
0.50	**20**	384.6	7693	926	0.120	**20**	204.2	4083	48	0.012
0.60	**20**	282.6	5651	140	0.025	**20**	203.3	4067	58	0.014
0.70	**20**	263.2	5264	174	0.033	**20**	204.2	4083	63	0.015
0.80	**20**	343.6	6873	948	0.138	**20**	204.4	4089	72	0.018
0.90	**20**	334.9	6698	947	0.141	**20**	205.7	4114	**96**	**0.023**
1.00	**20**	327.2	6544	45	0.007	**20**	**331.0**	**6620**	44	0.007
				`Depot`						
0.00	4	453.2	1813	39	0.022	4	473.2	1893	33	0.017
0.10	6	308.0	1848	58	0.031	6	315.2	1891	39	0.021
0.20	8	232.2	1858	57	0.031	8	236.9	1895	45	0.024
0.30	7	141.0	987	52	**0.053**	8	232.6	1861	43	0.023
0.40	8	233.6	1869	71	0.038	9	208.7	1878	37	0.020
0.50	**10**	187.4	1874	70	0.037	**10**	189.4	1894	48	0.025
0.60	8	232.4	1859	55	0.030	9	209.4	1885	38	0.020
0.70	9	206.2	1856	57	0.031	9	210.1	1891	40	0.021
0.80	7	270.6	1894	101	**0.053**	8	236.6	1893	38	0.020
0.90	8	238.4	1907	**118**	0.062	8	239.4	1915	**58**	**0.030**
1.00	8	**730.9**	**5847**	24	0.004	9	**631.9**	**5687**	31	0.005
				`Driverlog`						
0.00	7	**790.6**	5534	1344	0.243	7	306.6	2146	81	0.038
0.10	9	574.6	5171	978	0.189	9	239.2	2153	**96**	**0.045**
0.20	12	462.1	5545	1446	0.261	12	178.8	2145	87	0.041
0.30	12	474.2	5690	**1511**	0.266	12	179.3	2152	88	0.041
0.40	12	468.9	5627	1460	0.259	12	180.3	2164	89	0.041
0.50	12	475.5	5706	1478	0.259	12	181.2	2174	93	0.043
0.60	12	439.6	5275	1028	0.195	11	195.3	2148	88	0.041
0.70	12	423.3	5080	850	0.167	12	180.2	2163	77	0.036
0.80	**13**	391.4	5088	879	0.173	**13**	166.2	2161	81	0.037
0.90	**13**	456.6	5936	1803	**0.304**	**13**	166.6	2166	80	0.037
1.00	**13**	622.8	**8097**	45	0.006	**13**	**623.2**	**8102**	45	0.006

One final confounding factor that still needs to be addressed is that all WA* algorithms use a weighted diversity heuristic. It is not clear how the scale of the diversity value to the heuristic value impacted these findings. We ran an experiment using scaled values and found that using the raw diversity values dominated using scaled values. Still, the raw diversity values grow proportionally with the number of solutions found while the heuristic values from h do not change with respect to the plan set. It could be that later solutions are hard to find because h_{new} becomes dominated by the $h_{diversity}$ component. The results in Table 4.7 suggest a diversity-dominated h_{new} (i.e., lower values of *alpha*) would perform poorly. It also suggests that varying alpha during search may improve the results.

4.9 The Tradeoffs of Generating Plan Sets

Tradeoffs are inherently about understanding the interactions between one or more axes of evaluation, which for this study are efficiency, quality, and diversity. Understanding these interactions justifies specific algorithm designs and tells us where to set appropriate computational bounds. For example, we may wish to invest more search effort if doing so will lead to better solutions. Or, we may be able to adjust algorithms if we identify a tradeoff along other axis.

To examine whether more search effort results in improved quality or diversity, we partition the plan sets of the unique plans produced by each algorithm into intervals $i \in \{10, 25, 50, 100, 250, 500, 1000\}$. For each interval, we take the first i *unique* plans from each algorithm and call this the cumulative unique solutions at level i. In other words, we drop any duplicates until we reach i unique plans. We will discuss the `Transport` domain in detail since it has the most plans across all algorithms. We don't have enough unique solutions from the algorithmic approaches to draw fair conclusions for the three IPC-2002 problems, `Depot`, `Driverlog`, and `Rover`. Similarly, while the number of unique solutions for `Cybersec` is higher, few approaches create more than a 200 unique solutions. Out of the 30 problems, *Div* alone achieves this four times while *ITA* and *Hybrid* together achieve this ten times. These low counts of unique solutions further limit analysis of long-term trending.

So we draw our conclusions from the `Transport` domain with the caveat that these findings may not always transfer to other domains.

Figure 4.6 plots one example from `Transport` problem 10 of diversity (top) and quality (bottom) over the i unique plans; the data for all 20 problems is shown in Figures B.2 thru Figures B.5. When we looked at diversity across the intervals for each algorithm, several trends emerge. First, *RWS* and *LPGd* produce more diverse sets for 18 of the 20 problems. In five problems *Div* is close to either *RWS* or *LPGd*. In four problems, the plan sets of *Div* are as diverse as those from *RWS* or *LPGd*. In the case of the example from Figure 4.6, *Div* (a circle in the figures) actually begins to produce lower diversity around 750 solutions, and has a dip in diversity after about 50 solutions. *RWS* (a diamond in both figures) produces higher diversity plan sets much of the time. But it sacrifices quality to do so (see bottom plot). Except for four problems, *RWS* and *LPGd* tend to establish this high diversity by the first ten plans and maintain it for the duration of the intervals. In nine of the problems, *Div* starts low but trends up past the 200th unique solution, though it often stops trending after that point.

In terms of quality, different trends are seen. the general trend across the `Transport` problems is that *LPGd* produces the best quality (i.e., the lowest quality), while *RWS* produces the worst and the algorithms tend to fall in the middle. In seven problems, the algorithms also dropped quickly in quality (usually by the 200th solution). In a different set of seven problems, the algorithms dropped slowly in quality. For all algorithms, the quality tends to hover around the initial quality of plans found or gets progressively worse. This is expected given that we are not bounding the solutions returned based on solutions we have already seen (i.e., as in using some kind of upper bound procedure within the A* approaches). These results confirm that considering quality alone when attempting to drive diversity is unlikely to be productive. They suggest that we must more carefully consider this tradeoff of quality and diversity.

We wondered how often high diversity is paired with poor quality. To assess this tradeoff, we compute the *average* quality and *average* diversity for the first 10 unique plans of each

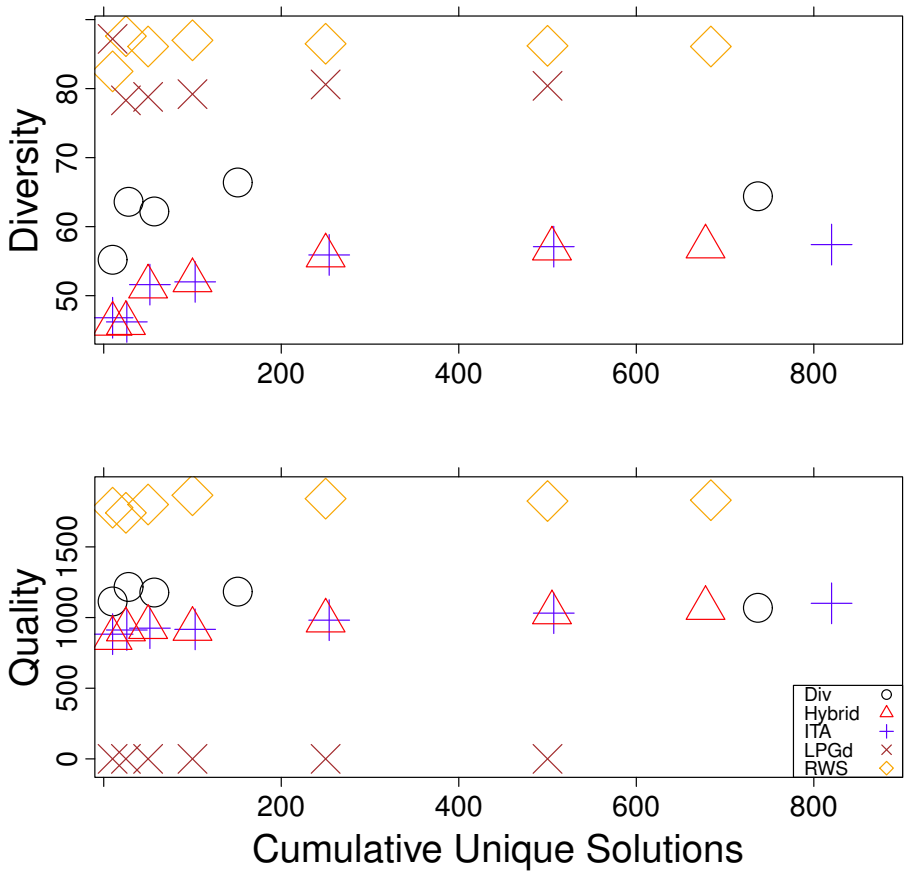

Figure 4.6: Diversity (top) and Quality (bottom) over the cumulative unique solutions $i \in \{10, 25, 50, 100, 250, 500, 1000\}$ for `Transport` problem 10.

algorithm (or fewer if the algorithm did not find 10 unique plans). Figure 4.7 plots the average quality and diversity for all problems from `Transport`. It is evident there is a strong tradeoff between high diversity and good quality. *RWS* exhibits a broad range of quality and diversity values, *Div* tends to cluster toward the origin and *Hybrid* and *ITA* fit in between with somewhat similar tradeoff. For problems solved by at least two algorithms, this trend was seen in 19 of 20 problems in `Transport` and all problems in `Depot`, `Driverlog`, and `Rover`. Figure 4.8 plots all the problems for `Cybersec`, where three distinct groups of quality each provide different diversity values and the algorithms have more even behavior across both dimensions. The solutions tend to be of much better (lower) quality though the diversity of those good quality solutions can vary considerably.

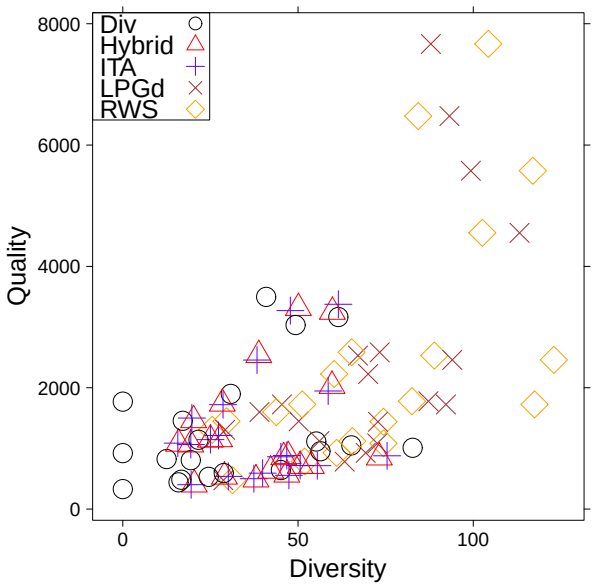

Figure 4.7: Demonstrating the tradeoff of quality (lower is better) and diversity (higher is better) in `Transport`.

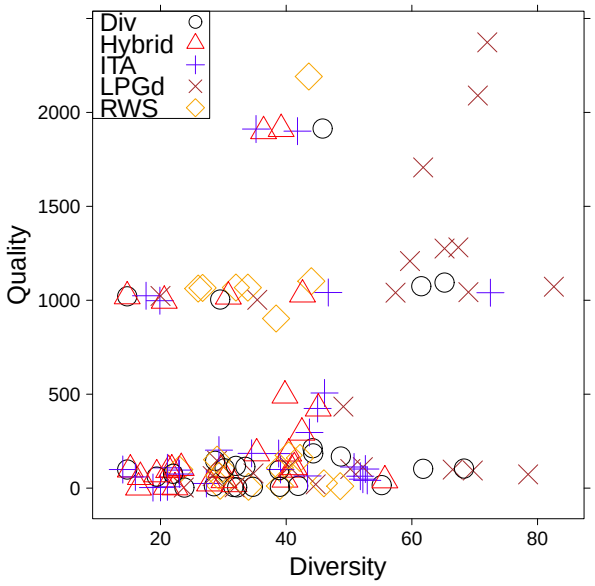

Figure 4.8: Demonstrating the tradeoff of quality (lower is better) and diversity (higher is better) in `Cybersec`.

4.10 Summary

Among the algorithmic approaches, we investigate tradeoffs in planner performance along the dimensions of quality, diversity, and efficiency. In terms of uniqueness, *RWS* can produce

many more unique plans than the other algorithms. Each algorithm produced unique plans not found by another. Solution quantity may be more important in applications (e.g., security), where one is seeking all possible plans. In applications where reducing solution overlap is important, one may want to use multiple approaches. In terms of diversity, *RWS* produces the most diverse plans in `Transport` while the WA* algorithms produce varying results for other problems. For search cost, the WA* algorithms perform similarly. *RWS* is inefficient and incurs a high initial cost and then produces a large number of plans. Finally, for plan quality, the *ITA* and *Hybrid* algorithms perform well and usually outperform the other algorithms.

When examining the full planning system *LPGd*, we found that it generally solved more problems and generated many more unique plans with higher diversity than the algorithmic approaches. It also appears to generate unique plans for every solution (its ratio is consistently near 1). But, when we examine the plans produced by *LPGd*, it is clear that it generates artificially unique plans by including numerous extraneous actions and very long plans. This has a side effect of also artificially boosting the diversity value for *LPGd*, which in some domains such as `Transport` can lead to lower solution quality. This alone does not single out *LPGd* as any worse than other approaches, but rather it makes a fair comparison difficult under the metrics we used in the study.

Investing more search effort does not result in plans that get better on average in an anytime fashion. *Div* produces unique plans early in search but tends to taper very quickly and starts producing many duplicate plans. *Hybrid* and *Div* both produce good quality plans at the start but then hover around the initial quality they find. *RWS* tends to stagnate. *LPGd* produces very good quality early on but does not explore much of the diversity-quality tradeoff.

The broader implication of these findings is that researchers and practitioners should carefully consider the needs of the application. Paradoxically, using only diversity to guide search does not always produce the most diverse plans according to an action-based diversity metric. Embedding diversity in the heuristic can result in a large number of duplicated plans.

Pairing a state-based Tabu list with a diversity heuristic in a hybrid algorithm is inefficient, largely because the weighted objective function approach misleads search and generates many duplicates that are either produced as plans or must be skipped.

This chapter presents several contributions to the existing literature on generating plan sets. We compare several different approaches for generating a plan set. As far as we are aware, this is the first work to join these multiple approaches into a single evaluative focus. We show that each approach produces plans not found by the other, which suggests that each algorithm has distinct value for generating alternatives. Our study also shows that all approaches produce their most useful results early regardless of the metric used. Thus, long searches are not likely to be productive – this is an important observation when planners are placed in the loop of an executing or mixed-initiative system. We demonstrate that *ITA*, which was designed as part of this dissertation, is best suited to the security domain for which it was originally designed. Since the security domain does not require such a strong focus on the length of plans at the expense of other concerns, it seems that it is the kind of application where search needs to be driven by something other than plan length. Overall, these results reveal a tradeoff between diversity and quality. From a design standpoint, deciding which is more important to an application is the first step in developing or tailoring an algorithm for generating alternatives.

A key understanding is that using diversity to both drive a planner's search *and* simultaneously evaluate its performance may be misleading in a larger context of all three dimensions of quality, diversity, and efficiency. Doing so can lead to plans that violate parsimony by containing extra action sequences which a human or agent could never tolerate. It can lead to inefficient search by skipping duplicate solutions (i.e., low uniqueness) rather than focusing search on creating novel solutions. Best First Search can be misled by what we call a diversity-quality tradeoff if plan cost and diversity are paired together to drive search. We usually want high values for diversity but low values for quality, and yet, quality and diversity can interact in subtle, sometimes contradictory ways. Finally, the issue is further complicated when diversity is used within a weighted heuristic, as is done in both *Div* and

LPGd. An imbalance between diversity values and heuristic values (i.e., plan length or plan cost estimates) can seriously hamper search if diversity overtakes the original heuristic. In the next chapter, we examine the extent to which such metric interactions would impact search behavior, while continuing our analysis of the tradeoffs of incorporating diversity into search.

Chapter 5
Understanding Metric Interaction[11]

The focus of many planning systems is generating solutions that minimize (or maximize) the solution according to a *single* metric or a weighted combination of metrics. This may be a reasonable approach when the metric(s) interact in an understood way and we are producing a single solution. But it is less clear how one could generate a plan to satisfy multiple metrics that interact in unexpected or competing ways. It is further less clear how to generate diverse plan sets while also handling multiple metrics. Recent work examining plan diversity [CMA11; Ngu+12] combines diversity metrics with the original metrics in a single objective function, which further obscures the role of the objective function with respect to the evaluation metrics for planner performance. In this chapter, we examine the impact of metric interaction when shifting between producing single plans and plan sets. Figure 5.1 captures this focus.

As motivated in Chapter 2, many applications cannot be reduced to a single, weighted metric for quality. The search for plans in such domains requires a broader view of how the plans may tradeoff one quality metric for another. For example, in the motivating security domain, we are interested in finding diverse alternatives when quality metrics such as the likelihood of an attack and the cost of an attack may interact in subtle ways or not at all. Comparing plans across two or more metrics in such applications is challenging because the metrics may be competing or incongruous. For example, the makespan of a plan might be lower but the set of actions to achieve a lower makespan might use more costly resources. In the security domain, we may want to see paths leading to attacks that may be more detrimental or less probable than other attacks.

We looked to the existing planning literature on how to proceed with the potential metric interactions of security domain and discovered a gap. While there is a wealth of literature

[11]Portions of this work are from the paper: M. Roberts, A. Howe, I. Ray, *A Tale of 'T' Metrics*. The ICAPS-13 Workshop on Heuristics and Search for Domain Independent Planning, 2013, Rome, Italy.

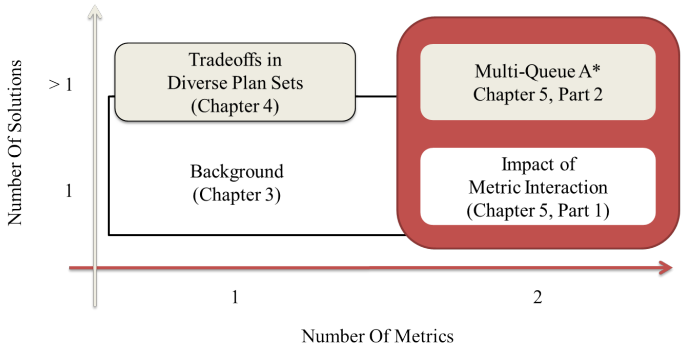

Figure 5.1: The two research directions of this dissertation with the focus of this chapter highlighted in red (dark gray).

exploring metric interaction within temporal domains (e.g., SAPA [DK03], LPG-td [GSS06], COLIN [Col+12]), the security domain does not require deep temporal or resource reasoning. The literature does not explore very deeply the issue of having multiple non-temporal metrics, and the existing benchmarks have few interactions between metrics for the small set of benchmarks that do have multiple metrics. Thus, there is a gap in research on numeric-only metric (non-temporal) domains, and this chapter presents the first results toward bridging that gap. Our findings complement the temporal+metric planning literature by probing deeper into the planner performance in the face of interacting non-temporal metrics. We discuss how metrics are used in existing IPC benchmarks and show that these domains rarely explore interactions between the metrics *within the same action*. The existing benchmarks have few, if any, of the critical interactions we expect to see in the security domain. To study the problem more deeply, we create a Bicriteria Synthetic domain (`BiSyn`) that allows us to vary the interactions of two criteria in a controlled manner while fixing plan length.

To examine search behavior for producing single solutions in a multi-metric problem, we run an off-the-shelf metric planner, MetricFF, on the synthetic problem. MetricFF implements a variant of A* called A*-epsilon (A^*_ϵ) that provides bounds on the plan cost of solutions it finds (see Background, Section 3.2 for details). We examine search cost and the ability to find minimal solutions for A^*_ϵ [PK82], as implemented in MetricFF [Hof03]. We find that this implementation of A^*_ϵ has the most difficulty finding minimal solutions

for collinear and random interactions, while it is most successful for simple curvilinear interactions that are strongly correlated with plan length. Uniformly scaling the metrics so that they less strongly correlate with plan length decreases the ability of A^*_ϵ to find minimal solutions. Weighting one metric more heavily than the other degrades search performance as well. These findings, though limited, provide strong evidence against combining the metrics in a weighted function if the metrics can be collinear or if they scale differently.

Continuing with our analysis of how metric interaction interplays with plan set diversity, we move in the direction of producing plan sets *within the bicriteria setting of* **`BiSyn`** . Recall that we show in the last chapter that driving search by adding diversity as a component of the heuristic function leads to inefficiencies because high diversity can imply poor quality. Instead, we seek to drive search by both quality metrics and the diversity metrics. In Section 5.3.2, we introduce an algorithm, Multi-Queue A* (MQA), that leverages an observation underlying the recent successes of several planners that use multiple heuristics (e.g., LAMA [RW10], Fast Downward [Hel06]). Since heuristics can be inaccurate and mislead search, such planners apply several heuristics during search, keep each heuristic in a distinct open list, and select across the set of open lists in some principled manner (usually round robin). Similarly, MQA maintains separate *queues*, $Q_1, Q_2, .., Q_t$, for each individual metric $t \in T$; we call them queues because they are more general than the usual A* or WA* open list. A queue can minimize or maximize a single metric or it can combine more than one metric (or heuristic) into more complex sorting schemes, as we will show when we create a queue to encourage parsimony. Further, we can also add the `(state, operator)` Tabu list from ITA, creating what we call MQA_T. We show that MQA can lead to different solutions depending on the queues it uses. However, the solutions it produces are still not very diverse across the spectrum of the tradeoff between quality metrics.

In a final linking of quality and diversity, we add two different kinds of diversity queues. The first queue, Q_D, where $D = D_{\text{stability}}$, leads to two variants MQA_D and MQA_{TD} and reveals that diversity alone is still inefficient for producing plans that are distinct from those of MQA and MQA_T (i.e., the MQA algorithm run without Q_D). This led to reconsidering

how to include parsimony as a key metric for generating diverse plan sets. As we point out in our discussion of the parsimony ratio (see Section 4.1), a planner which produces a shorter plan while maximizing diversity avoids the poor tradeoff between quality, efficiency, and diversity, as we demonstrate in Chapter 4. We show that, for a given level of diversity, the most parsimonious solution is that which minimizes plan length and that to do this implies minimizing h_{relax} first and – for all partial solutions at that level of h – then maximizing $D_{\text{stability}}$. Thus, we arrive at what we call a *parsimonious queue*, Q_S, with two final variants MQA_S and MQA_{TS}. We show that MQA_{TS}, in particular, produces highly diverse plan sets for `BiSyn`. Further, MQA_{TS} is competitive on the benchmarks from Chapter 4 and is very well suited to one benchmark domain, Depots, for which it produces the most unique plans. It finds the parsimonious plans it was designed to find.

Our results and analyses show that parsimony must be a central component of the planner design when considering how to generate diverse plan sets. We show that the most parsimonious plans are those that minimize the distance-to-go, which is usually calculated with reachability analysis, namely the relaxed graph plan. We embed this observation within *MQA* using a parsimonious queue, Q_S, and show that the approach leads to improved diversity for several domains. While Chapter 4 shows that maximizing diversity alone leads to inefficiencies, the results from this chapter unify and validate our new perspective of plan sets and multi-metric planning.

5.1 Evaluation Metrics

We employ the same evaluation axes as we used in Chapter 4: coverage, quality, efficiency, and diversity. Recall that coverage is the number of problems solved out of the set given and quality is measured as either *plan length* or *plan cost* (i.e., the sum of action costs). Efficiency is measured in terms of *CPU time* if planner implementations differ or *search cost* (i.e., the number of search nodes generated) if the implementations rely on the same framework. In some cases, we show box-and-whisker plots of CPU time to a completed plan. The diversity of a plan set is measured in several ways. We use the stability distance metric from the

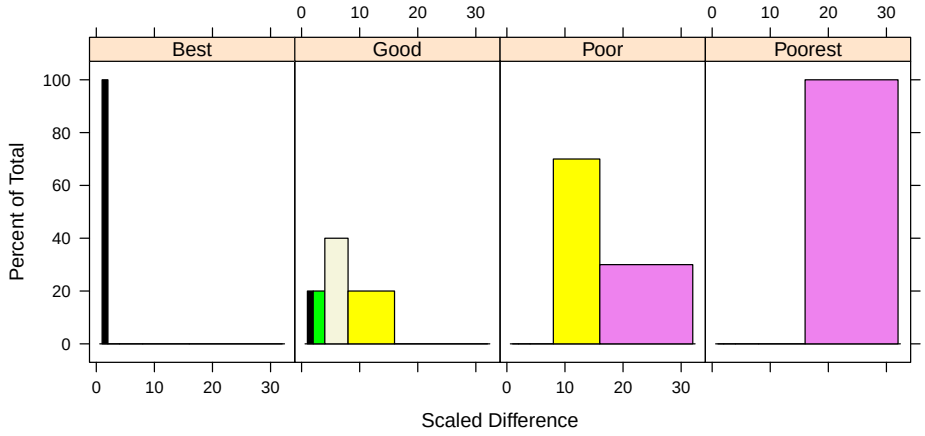

Figure 5.2: An example histogram explaining how to read later plots. Thinner bars to the left are best. Wider bars to the right indicate poorer performance. The y-axis is the percent of total problems (out of 30) that fall into each bin.

literature $D_{\text{stability}}$ (Equation 3.3), and contribute three additional metrics of *uniqueness* (u, Equation 4.1), the *parsimony ratio* (s, Equation 4.2), and *overlap* (o, Equation 4.3).

Since we are taking a new focus on plan quality, we add one additional quality metric for `BiSyn`. For plan cost on `BiSyn`, we state an algorithm ϵ-minimizes a function if most of its solutions fall within 8 times the optimal; this allows a slightly more relaxed requirement because we use $\epsilon = 5$ so there should be a guarantee that solutions are within a factor of 5 of the optimal solution. To plot how well an approach minimizes the plan, each plan, π, is first scaled, $cost(\pi)/cost*$, where $cost*$ is the minimal solution found using a simple dynamic programming algorithm. A minimal plan receives a 1; other solutions are factors of how much worse they are from the minimal solution. We then histogram these scaled values using bin sizes of $1, 2, 4, 8, 16,$ and $>= 32$. Figure 5.2 shows a comparative example of what we want to see for optimal and worsening solution quality. Thicker bars indicate higher variance, while the right-most bars indicate the poorest relative performance. Note that the y-axis in these plots is the percent of total problems, which for each function/metric combination is 30 problems.

All experiments are run on a 2.7Ghz Quad Core i7 with a time limit of 15 minutes and memory limit of 1GB.

5.2 Domains

Like the metrics, we use the same benchmark domains as Chapter 4, except that we do not include `psysec`. However, our work is still motivated by what we hope to include in the security domain. These included `Driverlog`, `Depot`, and `Rover` from IPC-2002, `Cybersec` from IPC-2008, and `Transport` from IPC-2011. We add a synthetic domain that we discuss next.

5.2.1 Existing Benchmarks in Planning

Applications with multiple, competing quality metrics (e.g., minimizing time versus money) are common but have been simplified in planning research to avoid explicitly reasoning about the trade-offs. The planning community has begun to look at more complex quality metrics, e.g., the net-benefit track in the 2006 planning competition [GL05] and the evaluation of planners based on plan cost in recent IPCs. While these have taken steps toward including more complex metrics and capturing the trade-offs, they remain focused on a weighted evaluation function that may not be appropriate for applications where the metrics interact.

Our security domain presents at least four metrics that are not readily combined into a single objective function. The *likelihood of attack* and *cost of attack* characterize the risk associated with doing nothing about a potential threat, while *utility to the user* and *cost of intervening* characterize the reward of better securing the system while minimizing user irritation. Searching for breaches in the security domain is unique in that a property called *monotonicity* [AWK02] eliminates cycles and bounds the length of plans, but the interacting metrics create a challenge for search because they may interact in subtle ways. For example, phishing attacks have a high privacy violation cost because a novice user may provide financial or personal details to a malicious third party, thus subjecting themselves to unpleasant consequences. Since such attacks can only happen through email, a low cost intervention might be to disallow email. However, this is unlikely to be a desirable intervention from the user's perspective given the ubiquity of email use for communication. A higher-cost intervention could be educating the user, but failing to do this well may increase the risk or

cause the user to be overconfident in their ability to assess a new threat. The agent of this security application must reason about a set of alternative plans which balance the complex trade-offs of how likely an attack will lead to a compromised system, of the costs associated with intervening against such attacks, and of the repair costs for a (potentially) compromised system. In short, there is no single, best solution. The security application is one example of metric interaction.

In other multi-metric applications, the metrics may have competing or subtle interactions that occur within an action. Another example is an autonomous system that needs to balance the overall objectives of a mission while assessing metrics such as power consumption, remaining time, risk and safety, level of required coordination, communications delay, etc. Another example is a human travel agent using a mixed-initiative planning system to book an itinerary that balances cost, total travel time, airport preferences, choice of seating, etc.

Existing multi-metric benchmark domains provide some inspiration in how to approach the multi-metric security domain. We highlight six non-temporal benchmark domains from IPC-2002 [LF03] that are represented in PDDL 2.1, Level 2 [FL03]: Depots, DriverLog, Satellite, Settlers, Rovers, ZenoTravel. These domains were a significant advancement toward metric planning and include at least two unique metrics in addition to plan length. Radzi [Rad11] shows that the single, weighted-sum metric in most of these domains interacts with plan length in ways that make the problem *straightforward* to solve. More specifically, Radzi distinguishes metric straightforwardness into a spectrum of strictly straightforward (plan length is a proxy for the metric), straightforward (plan length may be a proxy), semi-straightforward (plan length and the metric may diverge but the metric is monotonic in the plan length), and expressive (the metric may contradict plan length). Radzi examines several new domains for the semi-straightforward category and leaves to future work the set of expressive domains, of which we believe the security domain may be one case. Only Settlers has metrics that interact with plan length in an uncorrelated way, although the metric interactions are still constrained by a producer–consumer model. Radzi further shows that, for most domains, the metrics interact within a set of repeatable action sequences (e.g., in satellite,

turning the camera to take an image) or the metrics interact within a producer–consumer model where one action increases the availability of a resource that is later consumed by another action. While the metric benchmarks Radzi studied have a variety of metrics that do interact in constrained ways, the metrics rarely contradict plan length or interact *within* the same action.

5.2.2 Controlling for Metric Interaction in a Synthetic Domain

Drawing on the security domain as inspiration, we seek a domain where many solutions in the domain have similar plan length and cost between the metrics can interact. The monotonicity property of the security domain motivates a restriction that there cannot be cycles nor negative effects [AWK02] (see Motivation, Section 2.1.1 for details on monotonicity). We discuss this limitation in more detail in Section 5.7

Our solution is to create a synthetic domain where all solutions have the same plan length and the metrics are systematically varied across the operators of the domain. We focus on varying the interactions of two metrics, x and y, with each other. Figure 5.3 presents a pictorial view of the synthetic domain, which is essentially a fully connected planning graph. Nodes in this graph are states and arcs are transitions (operators). The $m \times n$ graph has m layers with each layer containing n states. In planning terms, m determines the plan length of any valid solution, while n determines the granularity of the metrics as discussed below.

States are labeled $s_{ij} \in \mathcal{S}$, where $i = \{0, 1, .., m\}$ indicates the layer, and $j = \{0, 1, .., (n-1)\}$ indicates one of the states within a layer. The initial ($i = 0$) and goal ($i = m$) layers have only one state; these are respectively labeled 'Init' and 'Goal' in Figure 5.3. Transitions between states are ordered pairs $(s_{ij}, s_{i'j'}), i < i', j < j'$ and labeled $a_{(ij,i'j')} \in \mathcal{A}$. The middle layers of the graph are fully connected with the next layer, so there is a path from any state in level i to all states at the next level ($i' = i+1, j' = \{0, 1, .., (n-1)\}$). There are $(m-2)n^2 + 2m$ transitions. Each arc in the graph represents a single operator, where the precondition links to a state at level i and the postcondition links to a state at level $i+1$.

Adding Cost Metrics To The Synthetic Domain We apply a weight matrix, W, over the transition matrix. Table 5.1 shows the weights of the transition matrix for the 3×3 problem. The weight of an arc is set to the value of the node it leads into (shown as a small integer at the top of the states in Figure 5.3). The arcs leading into the left-most states are all set to 0; formally, $w(a_{ij}) \rightarrow 0, \forall i, j = 0$. Arcs leading into the right-most states are set to 1000; formally, $w(a_{ij}) \rightarrow 1000, \forall i, j = (n-1)$. Arcs in the middle interpolate the distance between 0 and 1000. Formally, the values depend on the jth column: $w(a_{ij}) \rightarrow j * 1000/(n-1), \forall ij, 0 < i < m, 0 \leq j < n$. Thus, the number of states across the graph, n, determines the granularity of the metrics.

To weight this graph with costs, we apply two functions, x and y, which will become the metrics by which we evaluate plan quality. A third function provides $z = x + y$. The three metrics x, y and z are functions of W, the weighted transition matrix; we only set the value of x or y if a transition exists. The ranges of all metrics are integers $[1, 1000]$ to ensure some numeric effect in every action. Values of x are obtained: $x_{(ij,i'j')} = max(1, w_{(ij,i'j')})$. For brevity, we drop the subscripts and say $x = max(1, w)$. To obtain y, we apply functions that control the interaction between x and y. These functions are listed in Figure 5.4 and plotted in Figure 5.5. The random (ran) function is a no-interaction baseline. The other functions vary the interaction with functions (and their "mirrored inverses"): linear, lin (nil), sigmoidal, sig (gis), and polynomial, pol (lop). To aid in reading, note that the letters are reversed for each mirrored function. The polynomial function is rotated to provide a more interesting optimization metric. The fractional component of any function is truncated, and random noise $\epsilon = round(\mathcal{N}(0, 10))$ is added to all functions to generate randomized problems. If the function or the random noise cause a negative value, it is set to 1.

We partition the functions into two sets. The "easy" functions $E = \{ran, lin, nil, sig\}$ show no real tradeoff between x and y or have many minimal solutions (as in nil). The "difficult" functions $D = \{gis, pol, lop\}$ exhibit a trade-off with a small number of minimal solutions, although multiple symmetric solutions may exist because we discretize the functions. The ordering of the functions maintains the relative order of these partitions.

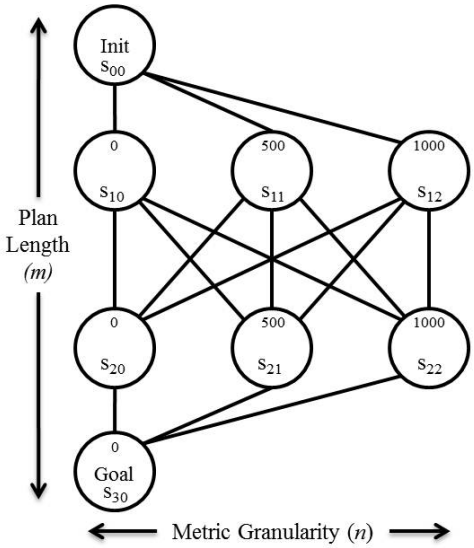

Figure 5.3: A graph of the 3×3 synthetic domain. Transition weights are at the top of the node an arc leads into. The graph is directed, and arcs only transition from the top down.

Table 5.1: The transition matrix for 3×3.

	1,0	1,1	1,2	2,0	2,1	2,2	3,0
0,0	0	500	1000				
1,0				0	500	1000	
1,1				0	500	1000	
1,2				0	500	1000	
2,0							0
2,1							500
2,2							1000

We apply the transition matrix and the values of x and y to create PDDL problems for a specific $m{\times}n$ size; this yields one domain file and 21 problem variants (i.e., seven functions, three metrics). Metrics are applied in the operator effects with `(increase (f) value)`. The problems are labeled $m{\times}n^{\text{metric}}_{\text{function}}$, where the function is one of $\{ran, lin, nil, sig, gis, pol, lop\}$ and the metric is one of $\{x, y, z\}$. Figure 5.6 shows the PDDL for $4{\times}3^{x}_{nil}$. The full listing for this domain is in Figure C.1.

$$x = max(1, w) + \epsilon$$

$$ran(w) = c \cdot uniform(1, 1000) + \epsilon$$

$$\begin{aligned} lin(w) &= x = max(1, w) + \epsilon \\ nil(w) &= max(1, 1000 - w) + \epsilon \\ sig(w) &= round(1000 * (1/(1 + e^{-(w-500)/110}))) + \epsilon \end{aligned}$$

$$\begin{aligned} gis(w) &= round(1000 * (1/(1 + e^{(w-500)/110}))) + \epsilon \\ pol(w) &= round(1050 * (1/exp(w/10)^{0.05})) + \epsilon \\ lop(w) &= round(1000 * (1000^3 - w^3/1000^3)) + \epsilon \end{aligned}$$

Figure 5.4: The functions used to vary metric interaction. See Figure 5.5 for a plot of each function.

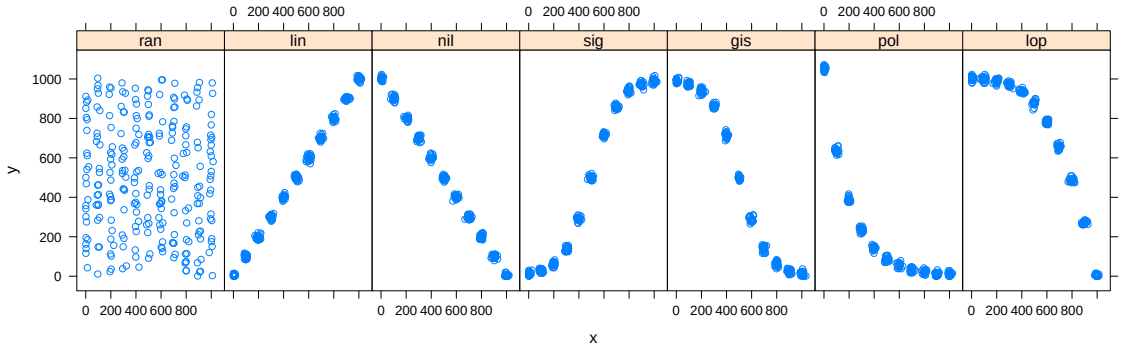

Figure 5.5: Examples of the interaction between metrics x and y.

```
(define (domain synthetic-4-3-nil)
  (:requirements :equality :typing :fluents)
  (:types State)  (:predicates (state-active ?s - State))
  (:functions (x) (y) )
  (:action Apply-00-10  :parameters (?state-00 - State)
   :precondition (and (state-active ?state-00)
                      (= ?state-00 State-00) )
   :effect (and (state-active State-10 )
                (increase (x) 1)
                (increase (y) 1000) ) )  ...)
(define (problem synthetic-4-3-nil-x)
  (:domain synthetic-4-3-nil)
  (:objects State-00 - State
            ...
            State-40 - State)
  (:init (state-active State-00) (= (x) 0) (= (y) 0))
  (:goal (state-active State-40))
  (:metric minimize (x) ) )
```

Figure 5.6: Partial PDDL for the domain and problem $4{\times}3^{x}_{nil}$; a full listing of this domain is included in Figure C.1

5.3 Implementations

We describe the two algorithmic approaches we apply in this chapter. The first is based on a metric planner, MetricFF [Hof03], and the second is our algorithm called Multi-Queue A* (MQA).

5.3.1 A-star-Epsilon (A^*_ϵ)

We used the A^*_ϵ algorithm as implemented in the newest version of MetricFF [Hof03]. Instead of popping the best solution ($s = min(q)$) from the top of the priority queue at each iteration, A^*_ϵ [PK82] selects a solution from the top K solutions (i.e., a focus list) on the top of the queue. A solution s' is in K if $f(s') <= 5f(s)$ and $h(s') = h(s)$. We only use MetricFF with cost optimization enabled. This implementation does not use helpful actions nor the initial enforced hill climbing stage. We use $m \times n = 15 \times 29$ because it was as large as MetricFF could still handle.

5.3.2 Multi-queue A* (*MQA*)

We now bring together several insights from across our study and present a new algorithm, Multi-queue A* (MQA), that offers one way to incorporate these insights in a variant of A*. The first issue is one of metric scale. In Section 5.4.1 (below) we will show that scaling the quality metrics can negatively impact search behavior. Thus, the need to manage metric scale between diversity values and heuristic values is a critical design insight. When combining diversity and multiple metrics, several issues conspire against the common practice of normalizing the metric values to managing differences in scale between them. When searching for plans, we do not have the luxury of knowing the maximum diversity and this quantity grows with each iteration. Nor may we always have reliable (heuristic estimates of) maximum values for the other quality metrics. The problem of normalizing diversity, quality, and heuristic values for use within a single objective function is further complicated because the heuristic is minimized while the diversity is maximized. Finally, even if we could get good estimates, these metrics may have unknown (or only partially known) ranges *during*

search, which makes it challenging to normalize them effectively to drive the search process.

Results from Section 4.9 demonstrate that quality and diversity can (and often do) interact. In Section 5.4.1 (below) we also show that metric interaction can negatively impact the plan quality of final plans. For the `BiSyn` domain we deliberately force interactions and observe some non-intuitive results. It seems clear enough that combining them in a weighted objective function, while simple to do, is likely to lead to pathological search behavior (i.e., getting stuck without a solution, inefficient runtime to solution, or poor quality solutions) at least some of the time.

Thus, the evidence we have collected suggests that we must manage these metrics independently from each other *during search* rather than try to combine them. In short, we must maintain independence of a set of metrics, T, that is itself composed of quality, heuristic, and diversity metrics. To accomplish this, we can draw inspiration from existing planners that manage separate heuristics in distinct open lists [Hel06; RW10]. The key idea behind Multi-queue A* (*MQA*) is to use separate queues, Q_t for each metric $t \in T$, and to sort each queue as is appropriate for that metric. We call them queues instead of open lists because, as we will show, not every queue sorts the nodes using the canonical A* f function. The use of a queue rather than an open list has implications for the completeness, optimality, and runtime guarantees of A*, and we conjecture that it may be possible to bound these in a manner similar to A^*_ϵ or *MQA*. If not, maybe *MQA* should really be called Multi-queue Best First Search. In Section 5.7, we will discuss these implications, among others, that are important future work for *MQA* and other multiple queue algorithms. For now, we maintain the A* name because some queues will still employ the f function from A*. This naming also follows a central theme of our work of applying directed changes to the A* algorithm rather than incrementally tweaking a full planning system.

Algorithm 4 shows the main outer function of *MQA*. It is very similar to *ITA* except that a set of queues, $\mathcal{Q}$, is created and passed into the graph search algorithm, shown in Algorithm 5. As in *ITA*, EXTRACT-STATE-ACTION-PAIRS() stores each $(state, operator)$ pair that is on the path to the solution into $\mathcal{T}$.This version of graph search is similar to the A*-

Algorithm 4 MULTI-QUEUE-A* ($\mathcal{P}$, numSolutions, maxSteps) returns a set of solutions or failure

```
1: solutions ← ∅
2: T ← ∅
3: Q ← SETUP-QUEUES( P )
4: while solutions.size() ≤ numSolutions do
5:    solution ← MULTI-QUEUE-GRAPH-SEARCH( P, T, Q, maxSteps )
6:    T ← T ∪ EXTRACT-STATE-ACTION-PAIRS( solution )
7: end while
```

Algorithm 5 MULTI-QUEUE-GRAPH-SEARCH ($\mathcal{P}$, $\mathcal{T}$, $\mathcal{Q}$, maxSteps) returns a solution or failure

```
1: closed ← ∅
2: Q.INSERT-INTO-ALL-QUEUES( MAKE-NODE( s_o ), Q )
3: stepsTaken ← 0
4: while stepsTaken ≤ maxSteps do
5:    node ← Q.removeNext()
6:    if isEmpty( Q ) then
7:       return failure
8:    end if
9:    state ← node.getState()
10:   if S_g ⊆ state then
11:      return SOLUTION( node )
12:   end if
13:   if not inClosedList( state ) then
14:      closed ← closed ∪ state
15:      Q.INSERT-INTO-ALL-QUEUES( EXPAND( node, P, T ) )
16:   end if
17:   stepsTaken++
18: end while
```

GRAPH-SEARCH (cf. Algorithm 1) except that the Open list, $\mathcal{O}$, is replaced by $\mathcal{Q}$, which is essentially a data structure holding the relevant queues and ensuring that the basic A* loop does not know about the set of queues. In other words, when `removeNext()` is called, $\mathcal{Q}$ takes care of incrementing the underlying queues as needed and simply returns the next node for A* to consider. Similarly, other updates to $\mathcal{Q}$ propagate to all queues in $\mathcal{Q}$.

There are two design decisions for *MQA*: how to set up the queues, as managed in the SETUP-QUEUES() function, and how to select across them during search, as managed by the REMOVE-NEXT() function. The SETUP-QUEUES() function takes care of setting up $\mathcal{Q}$ depending on the problem and configuration; we discuss later the specific queues used

in our experiments. We set up two types of queues for our study, which we explain using the metrics from the `BiSyn` domain as an example. Let t_x represent the quality metric x that we want to minimize. We create a queue Q_x that sorts according to the usual f function for A*. When we have a good estimate for h_x we can use $f_x = g_x + h_x$. When we do not, we can use the reachability heuristic h_{relax} (see Background, Section 3.2.1) to select the best action to apply and then estimate h_x using the cost of that new action; thus $f_x = g_x + \text{cost}(\text{bestAction}(\pi_{relax}))$. Note that using h_{relax} was designed to minimize plan length, and thus maintains parsimony with respect to producing diverse plans. But this design decision may be a poor choice if one is looking for longer solutions that may have a better quality (see Section 3.4.2 for a discussion of pathological cases for cost-based search).

A quality queue may employ a metric that is itself compound. For example, consider the $z = x + y$ metric from the `BiSyn` domain. Creating compound metrics may lead to improved search behavior if the metric interactions are well understood. But the queue, Q_z, that minimizes z is functionally the same from the perspective of *MQA*. While we have clear justification for combining a heuristic estimate with (compound) quality metrics, the case is not so clear for diversity.

To create diversity queues, we apply the same design principle of a distinct queue except that our results caution us to avoid combining diversity and quality (or its heuristic estimate) into a weighted function. Diversity must be sorted differently, and we propose that the guiding design principle should be parsimony. Recall that parsimony suggests we should prefer short plans while maximizing diversity (see Section 4.1). Let D_S be a distance metric that orders states first by minimizing h_{relax} (Section 3.2.1) and then breaks ties by maximizing $D_{\text{stability}}$ (Equation 3.3). We call this the *parsimonious diversity queue*, Q_S. As a baseline to understand whether parsimony is useful, we also consider a queue, Q_D that maximizes $D_{\text{stability}}$ alone. We expect to observe that employing Q_S produces more diverse plans that are also more parsimonious.

Now that we have discussed the queues, we must make the final design decision for *MQA*: a queue selection strategy that is managed by the REMOVE-NEXT() function. We showed

in previous work on a planning portfolio called BUS [RHF07] that a simple round-robin selection strategy is sufficient for many planning problems. So our implementation follows from this insight and similarly uses a round-robin strategy. We point out, however, that there may be application specific reasons for preferring some other selection strategy. We discuss the selection strategy as an open question for future work in Section 5.7.

We implement *MQA* in the Mosaic framework (see Appendix A) and provide six variants for this study. The variants depend on two choices: 1) whether the Tabu list of *ITA* is enabled, and 2) whether and which diversity queue is enabled. Table 5.2 lays out the six variants we use. Note that the Tabu list or a diversity queue are relevant only after the first solution is found. It does not make sense to enable either of them when searching for a single plan; thus, we only run the vanilla *MQA* when producing a single plan.

We decide which queues to use at runtime based on the problem being solved. Which queues are enabled is another independent variable in our experiments for `BiSyn`. Table 5.3 shows the problems we explore along with the quality queues we used. In the case of the `BiSyn` domain, we employ five different queue configurations, and it is always the case that the first letter of the name for a configuration states the evaluation used for the solutions. Two configurations evaluate the solutions according to x and y while using the same corresponding quality queue. The other three configurations evaluate the solutions according to z but vary the queues: one uses only z, the second (xy) uses x and y only, and the third (xyz) uses all three quality queues x, y, and z. In the benchmark domains, we employ a single quality queue but the quality metric changes depending on the domain: the quality metric is plan length in the older IPC-2002 problems of Depot, DriverLog and Rover and the quality metric is the sum of action costs in the newer IPC-2008 `Cybersec` and IPC-2012 `Transport` problems.

5.4 Results: Producing Single Solutions for `BiSyn`

We begin with a study of `BiSyn`, where we apply both algorithms to generate single solutions. In this section, we analyze the two approaches for generating single solutions.

Table 5.2: The variants of MQA. When generating plan sets, the Tabu list can be on or off, and is designated with a subscript T (e.g., MQA_T). Use of the diversity queue is designated with a D (e.g., MQA_D), while use of the parsimony queue is designated with an S (e.g., MQA_S). A dash indicates that this particular combination is unused.

	Diversity Queue		
	None	Q_D	Q_S
Tabu Off	-	MQA_D	MQA_S
Tabu On	MQA_T	MQA_{TD}	MQA_{TS}

Table 5.3: The queues used during experiments for MQA. Each run of MQA for a particular problem contained one or more quality queues. A 'D' denotes MQA runs using the diversity queue Q_D while 'S' denotes runs using the parsimony Q_S. A dash indicates that this particular combination is unused.

Domain (Name)	Quality Metric	Quality Queue(s)	Diversity Queue(s) MQA_T	MQA_D MQA_{TD}	MQA_S MQA_{TS}	Preferred Operators
`BiSyn` (x)	x	x		D	S	Off
`BiSyn` (y)	y	y		D	S	Off
`BiSyn` (z)	z	z		D	S	Off
`BiSyn` (zxy)	z	xy		D	S	Off
`BiSyn` ($zxyz$)	z	xyz		D	S	Off
`Depot`	$\|\pi\|$	f	-	D	S	On
`Driverlog`	$\|\pi\|$	f	-	D	S	On
`Rover`	$\|\pi\|$	f	-	D	S	On
`Cybersec`	$c(\pi)$	f	-	D	S	On
`Transport`	$c(\pi)$	f	-	D	S	On

Our goals in this section are to lay a foundation of understanding when and how metric interaction impacts search. Our measurements include search cost as well as the scaled histograms (see Figure 5.2). Where appropriate we back up our observations with statistical testing.

5.4.1 Single Solutions for `BiSyn` using A^*_ϵ

We explore the behavior of A^*_ϵ on `BiSyn`. We observe the impact of metric interaction, whether there is a difference between easy (E) and difficult (D) functions, and the extent to which scaling impacts search behavior.

Table 5.4: Significance, p-value, and statistics (average time and standard deviation) for the paired-sample t-test on the quality metrics x and y for the original problems (top) and uniformly scaled problems (bottom). A single asterisk ('*') indicates significance at the $p < 0.05$ level. Three asterisks ('***') indicate significance at the $p < 0.001$ level.

			x		y	
	Sig.	p-val	Avg. time	SD	Avg. time	SD
ran	***	0	1.41	0.101	4.48	0.958
lin		0.248	1.42	0.139	1.38	0.107
nil		0.669	1.41	0.109	1.42	0.088
sig	***	0	1.48	0.152	2.52	0.476
gis	*	0.034	1.43	0.104	3.99	6.28
pol	***	0	1.43	0.115	1.94	0.372
lop	***	0.001	1.47	0.148	1.36	0.083
ran	***	0	1.52	0.189	4.1	0.953
lin		0.984	1.54	0.169	1.54	0.203
nil		0.751	1.44	0.162	1.45	0.153
sig	***	0	1.37	0.174	2.03	0.297
gis	***	0	1.43	0.162	2.09	0.338
pol	***	0	1.4	0.185	1.82	0.332
lop	***	0	1.42	0.16	1.27	0.048

The Impact of Metric Interaction Many planners are designed to minimize a single objective function. So we expect to observe little performance difference when minimizing either x or y alone. We study whether, for all functions, A^*_ϵ ϵ-minimizes both x and y with insignificant differences in CPU time between the two metrics. Table 5.4 (top) shows the runtimes of x and y along with the p-values for a pairwise t-test between x and y. These are also plotted on a log scale in Figure 5.7 (top). All the runtimes between the two metrics were significant ($p < 0.0073$) except the linear functions[12]. Figure 5.8 (left) shows a histogram of the scaled differences for how well A^*_ϵ minimizes x and y. A^*_ϵ has trouble finding the ϵ-minimal solutions.

[12]The Bonferroni adjustment controls the experiment-wise error of 7 pairwise comparisons at $\alpha = 0.05$, so the critical value for p is $0.0073 = 1 - (1 - 0.05^{1/7})$.

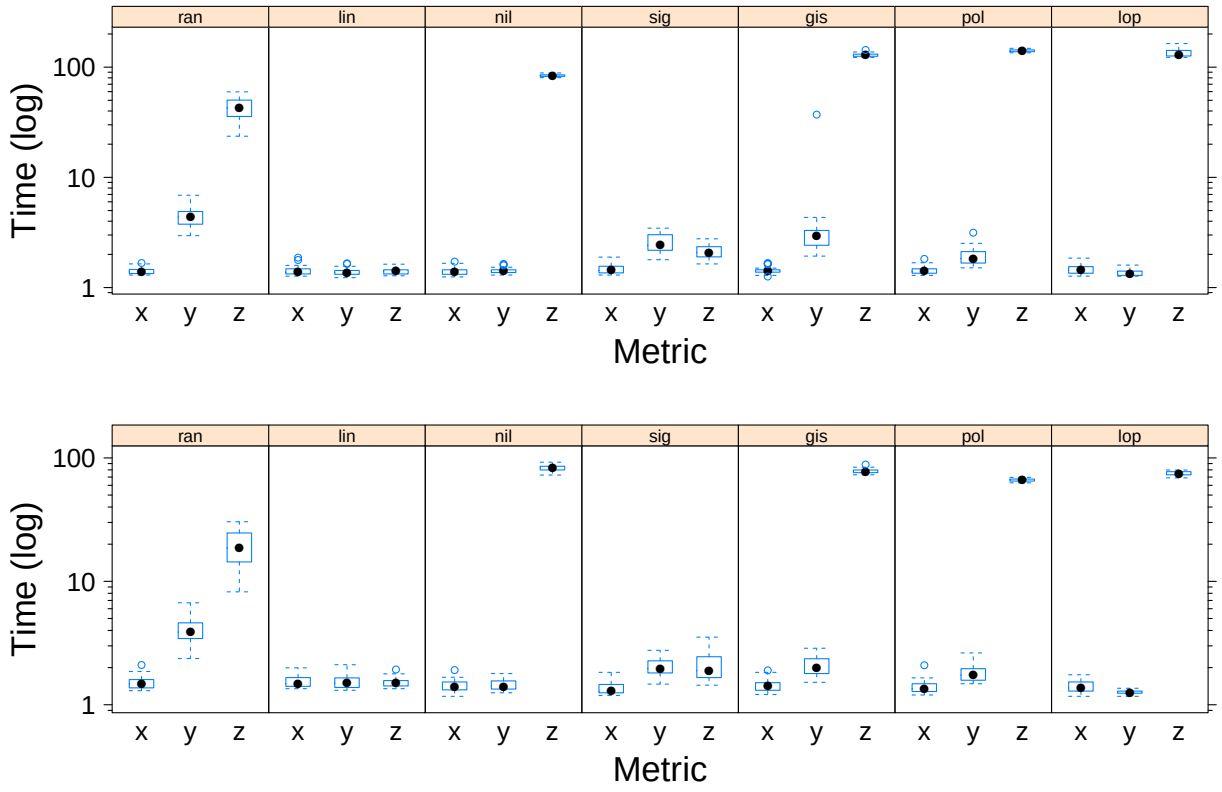

Figure 5.7: Runtime distributions of running A^*_ϵ all functions and metrics for the original problems (top) and the uniformly scaled problems (bottom).

There is a marked difference between minimizing x and minimizing y for some problems. Both metrics have a range that is represented equally in the actions at each layer of the graph. However, x is sampled at specific points that are interpolated between $[1, 1000]$ while y is sampled uniformly randomly in that same range. This sampling bias is evidenced in the vertical 'bands' for x seen in Figure 5.5 that are absent for y. This leads to more potential values for y than x, which appears to be more challenging for search. We plan to confirm this explanation in future work.

So we can conclude that, for the most part, ϵ-minimal solutions are not found and that there are significant differences in CPU time between minimizing x and y.

Comparing Easy and Difficult Functions For minimizing z, we expect to see quality and performance degrade as the xy tradeoff becomes more challenging. So we examine

planner performance on the easy and difficult metric interactions. The intuition behind this is that collinear functions are easy but non-collinear functions are more difficult. Finding the minimum should take longer and happen less frequently under the difficult interactions. We expect to see that, for the easy functions (ran, lin, nil, sig), A^*_ϵ will ϵ-minimize z and with insignificant differences between the CPU time of minimizing x, y, and z.

Figure 5.8 (left) shows that A^*_ϵ successfully finds ϵ-minimal solutions in z for nil, gis, pol, lop. In terms of runtime, Figure 5.7 (top) shows there is often a significant cost (10 to 100 times more) to minimizing z except in the simplest collinear functions lin and sig. The runtime for minimizing z is significantly different except for lin.

We also expect to see that for the difficult functions (gis,pol,lop), A^*_ϵ will not find the global minimums in z, and the runtime will be significantly different between D and E. Further we expect to see the difference in runtime will be similar between pol and lop, but the difference in runtime will be significantly different between gis and $\{pol, lop\}$.

Figure 5.7 (top) shows that the runtimes between the E and D problems are distinct, with the D problems usually taking more runtime. The runtimes between pol and lop, while overlapping, are statistically different; a Tukey HSD test run on the runtimes of both the E and D does not group any of the functions together. Figure 5.8 shows that A^*_ϵ successfully finds the minimal solutions for gis, pol, lop. So we conclude that CPU time increases for the difficult functions but there is not otherwise similar performance with the easy/difficult groupings. A^*_ϵ finds minimal solutions for all three functions.

The Impact of Scaling Several recent results have linked the sensitivity of search to the operator costs. Wilt and Ruml show that cost-based search is sensitive to the ratio of the operator costs [WR11]. Cushing et al. show that cost-based search can be misled by cost functions in the actions that work against heuristic distance [CBK11]. Finally, Sroka and Long [SL12] show that some planners are more sensitive to the metrics; for example they show that MetricFF (among other planners) can generate more diverse solutions by varying the constraintedness of resources in a logistics domain. We are interested in how the search

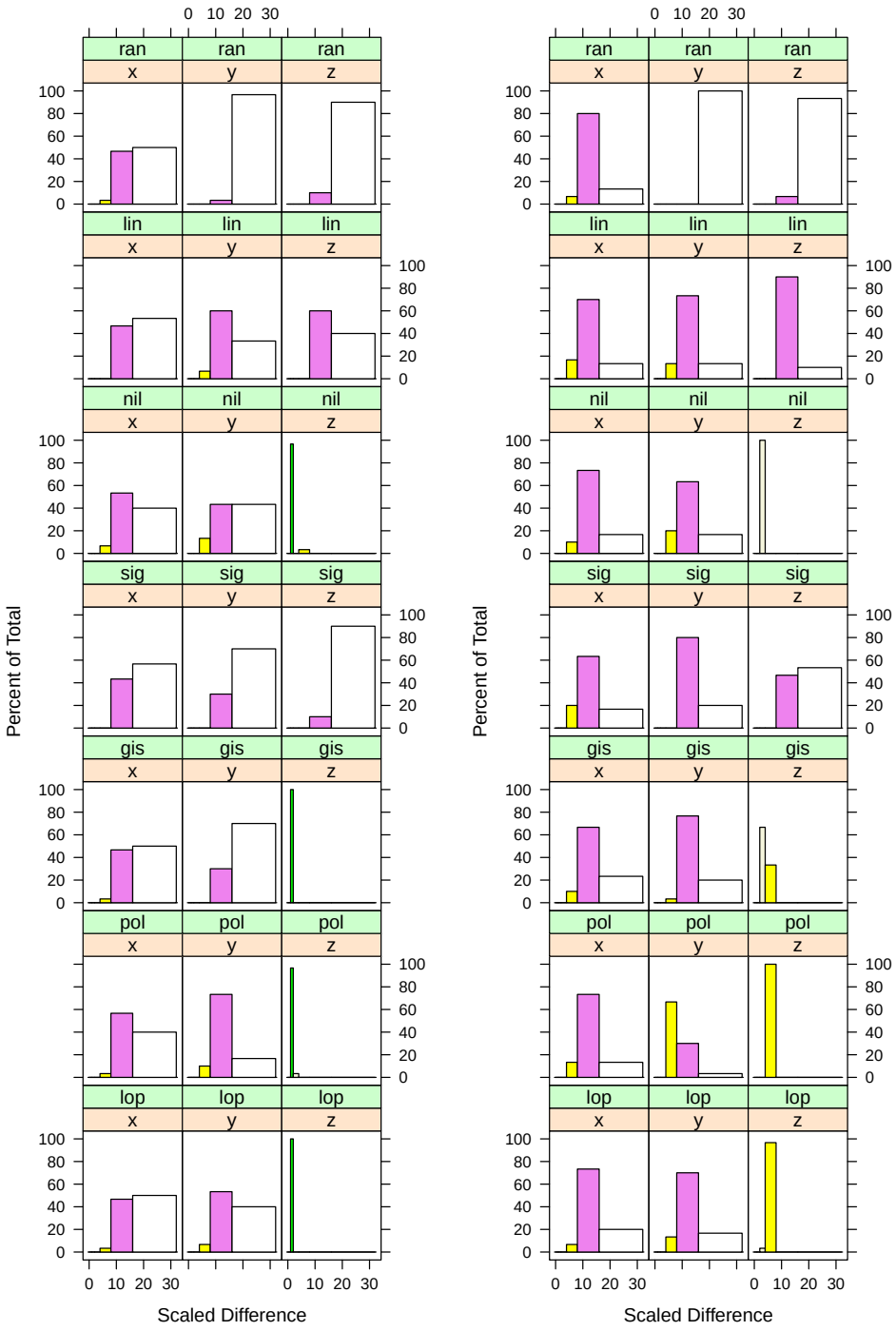

Figure 5.8: Log-Histograms of the scaled difference from the minimum for the original problems (left) and the uniformly scaled problems (right). Bins for the bottom axis are set at $\{0, 1, 2, 4, 8, 16, \geq 32\}$ to provide a visual representation of how well an algorithm does. Better performance is indicated by thinner and taller bars to the left. For more details, see the discussion of Figure 5.2.

behavior changes when we scale the values of the metrics uniformly or with respect to each other.

Our first set of experiments uniformly scales x and y together. We create a set of uniformly scaled problems that simply scale the original metric values in actions by 0.3. Figure 5.7 (bottom) and Table 5.4 show the runtime distributions for the uniformly scaled problems. The results parallel those of the correlated problems except for a more significant difference in *gis*, which can be explained by an outlier present in the original runs.

Figure 5.8 (right) shows histograms of the scaled difference from minimal solutions for the uniformly scaled problems. When minimizing x or y, A^*_ϵ finds equal or better solutions for the uniformly scaled problems, except for *ran*. In contrast, when minimizing z, A^*_ϵ finds worse solutions except for *lin* and *sig*.

We can also scale either x or y when minimizing z, which leads to a skewed evaluation function that favors one axis over another. We introduce functions that vary the weight of x and y.

$$z_{2x} = 2x + y \qquad z_{2y} = x + 2y$$

$$z_{5x} = 5x + y \qquad z_{5y} = x + 5y$$

$$z_{10x} = 10x + y \qquad z_{10y} = x + 10y$$

$$z_{25x} = 25x + y \qquad z_{25y} = x + 25y$$

$$z_{50x} = 50x + y \qquad z_{50y} = x + 50y$$

Figure 5.9 demonstrates that solutions tend to get worse as the scale increases (to the right) for scaling along x. This trend is less pronounced (or absent) in the random and collinear functions (*ran*, *lin*, and *sig*) while very evident in the remaining functions. The scaling is similar for y, as shown in Figure C.2. This evidence suggests that managing scaling is not as critical (but still useful) when metrics are collinear or uncorrelated but critical when they interact in more subtle ways.

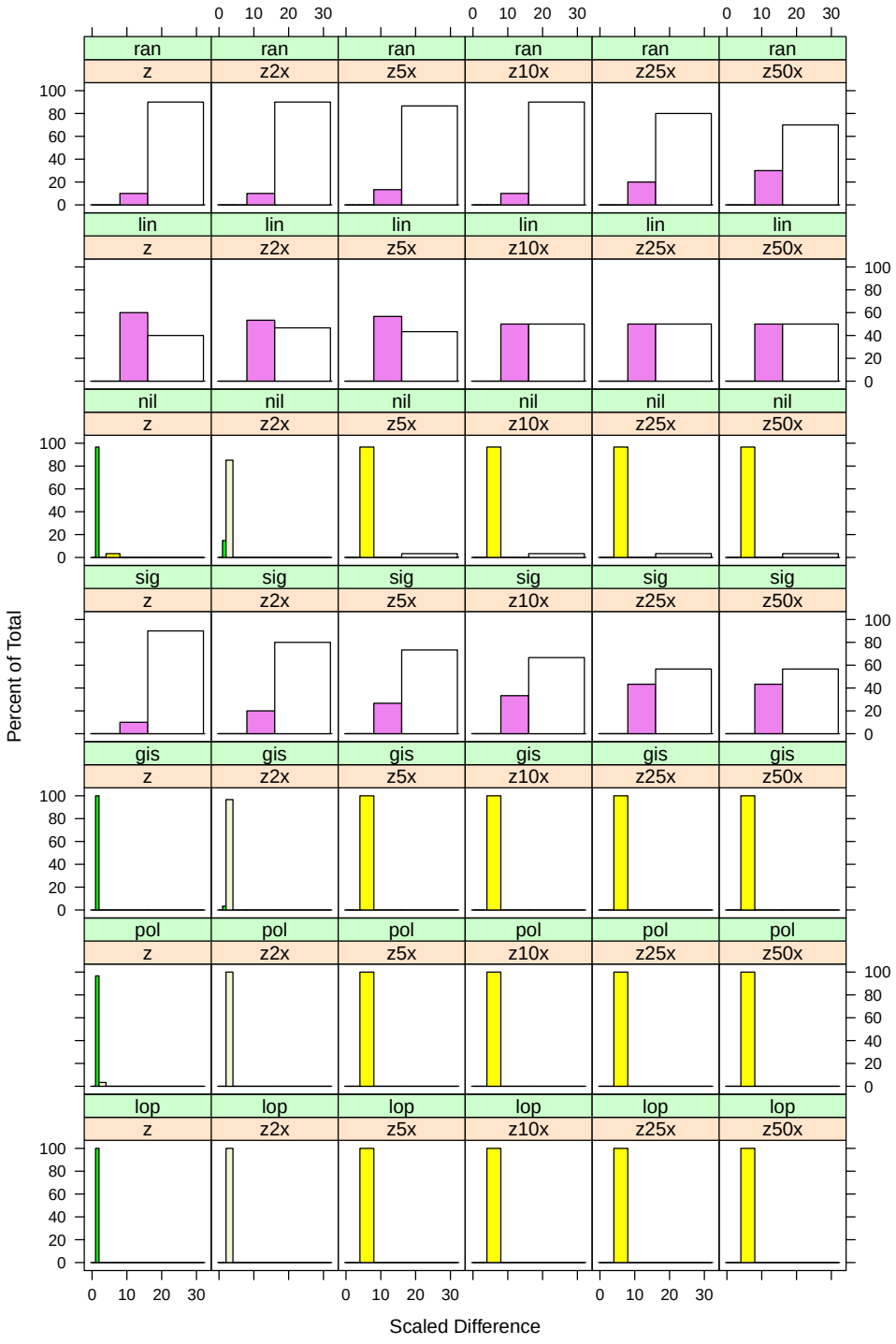

Figure 5.9: Log-Histograms of the scaled difference from the minimum for the z_{*x} problems. Bins for the bottom axis are set at $\{0, 1, 2, 4, 8, 16, \geq 32\}$ to provide a visual representation of how well an algorithm does. Better performance is indicated by thinner and taller bars to the left. For more details, see the discussion of Figure 5.2.

5.4.2 Single Solutions for BiSyn using *MQA*

In this section, we examine *MQA* on the BiSyn domain. Preferred operators are turned off for these runs because using a restricted neighborhood would impact the goal of the synthetic domain to assess how well the approaches use the alternative paths to construct solutions. We use several figures in this section to summarize the search cost (as runtime), the log-histograms, and the solution frontier of *MQA*, MQA_D, and MQA_S. Some of the larger plots would disrupt the flow of the prose and are placed in Appendix C; plots of the same type are placed on subsequent odd/even pages to allow for easier comparison.

Figure 5.10 shows the runtime histograms for *MQA* (top), MQA_D (middle), and MQA_S (bottom). It is evident that all *MQA* variants show a similar performance trend of larger search cost variance for the z function, which is similar to A^*_ϵ (cf. Figure 5.7). We performed a pairwise t-test between xy, xz, and yz while adjusting for experiment-wise error with a Bonferroni adjustment (i.e., a critical value for p of $0.0073 = 1 - (1 - 0.05^{1/7})$). For *MQA* and MQA_D, xy are always not significantly different, while xz and yz are always significantly different. For MQA_S, the same trend occurs except that xy is not significant for *pol* and *lop*, while, for xz and yz, all functions are significant except *sig*.

We also see that that *MQA* produces solutions with much less search effort than A^*_ϵ for all the functions except *lin* and *sig*. A Bonferroni adjusted t-test reveals that all versions of *MQA* are significantly faster to the first solution except for *lin* and *sig*. (cf. Figure 5.7). We believe that one explanation for this is the focus list of A^*_ϵ, *when focused on the quality metric*, can seriously mislead search. To see why, let us consider that the queue for A^*_ϵ is sorted by the quality metric f_t for a specific metric t. But for the BiSyn domain, quality and distance do not necessarily correlate with each other – in fact, we have deliberately caused plan length to be less interesting by forcing all solutions to be the same plan length. Yet, A* search is driven by both the cost of the current solution (i.e., its g-value) and simultaneously the cost of its heuristic estimate (i.e., its h-value). So the "best" action to achieve the next layer may be much deeper into the search queue than the size of the focus list. Alternatively, the focus list can grow very large, which decreases the chance that A^*_ϵ will even find that

"best" action. In such cases, it may be that h is a poor estimator to use. It may also be the case that A^*_ϵ is getting space restricted. Both of these possible explanations should be explored in future work.

Figure 5.11 provides a visual intuition for this effect using the *nil* function, where x and y are inversely correlated. The frontiers of Q_x (in orange in the upper left) or Q_y (in blue in upper right) are represented by large ovals over the states of a hypothetical `BiSyn` problem. Note that these two frontiers would be combined if we searched for z. A* is biased toward nodes with a low f-value, which equates to nodes with a low h-value (i.e., a better quality value) or a low g-value (i.e., a low cost-so-far value). Further, A^*_ϵ selects across this frontier – with equal likelihood – any node that is within the ϵ bound for the focus list. We believe that this is the key to understanding why A^*_ϵ is misled – expansions in this domain are very costly and A^*_ϵ expends lots of effort on nodes that are eventually shown to be less promising. This insight complements the existing literature showing that cost-based (i.e, quality-based) search can be misled in particular cases (e.g., [CBK11; WR11], see Section 3.4.2 of the background for more details). In particular, we extend those results to show that quality-based best-first search is misled when metrics interact with each other in specific ways and that search can be further misled when scaling occurs on top of the metric interaction.

It is not surprising that adding either of the diversity queues significantly increases the search cost over MQA without a diversity queue as diversity queues multiply search effort by a constant factor. However, the parsimony queue, Q_S, has slightly better runtime than Q_D. To understand why this might be the case, again refer to Figure 5.11. The hypothetical frontier for the parsimony queue, Q_S, is shown across the middle in a green oval (it has no filled background). Search moves one layer closer to the goal each time that MQA_S expands another node from Q_S. This gives MQA_S a slightly lower overall runtime than MQA_D even though using any form of diversity queue is still significantly higher than MQA alone.

In terms of achieving the optimal solutions, a glance over Figures C.3 and C.5 reveals that MQA and MQA_D tend to produce similar solution quality. When comparing this to the results from A^*_ϵ (cf. Figure 5.8, left plot) we can see that in addition to finding optimal

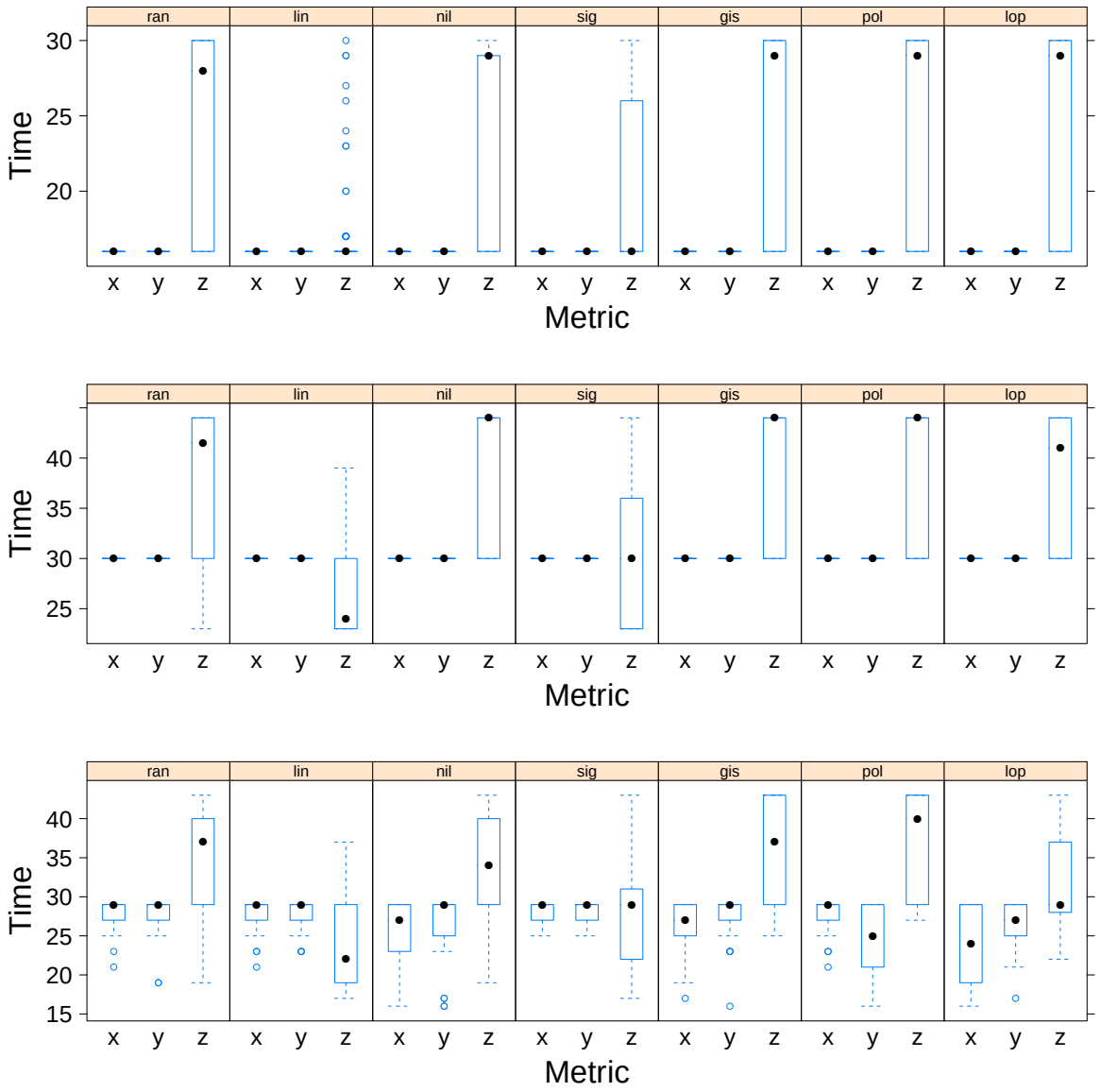

Figure 5.10: Runtime distributions of all functions and metrics for MQA (top), MQA_D (middle), and MQA_S (bottom) to generate a single solution in 30 problems.

solutions in the z function for *nil*, *gis*, *pol*, and *lop*, the two MQA variants tend to find better solutions than A^*_ϵ. This is in contrast to MQA_S as shown in Figure C.7, which has very poor solution quality. At first it might seem that the worse solution quality is undesirable, but recall that some domains exhibit a diversity-quality tradeoff where increased diversity implies decreased quality. So when we start considering how to increase diversity, we expect to see a drop in solution quality for those cases where the interaction between the two is correlated.

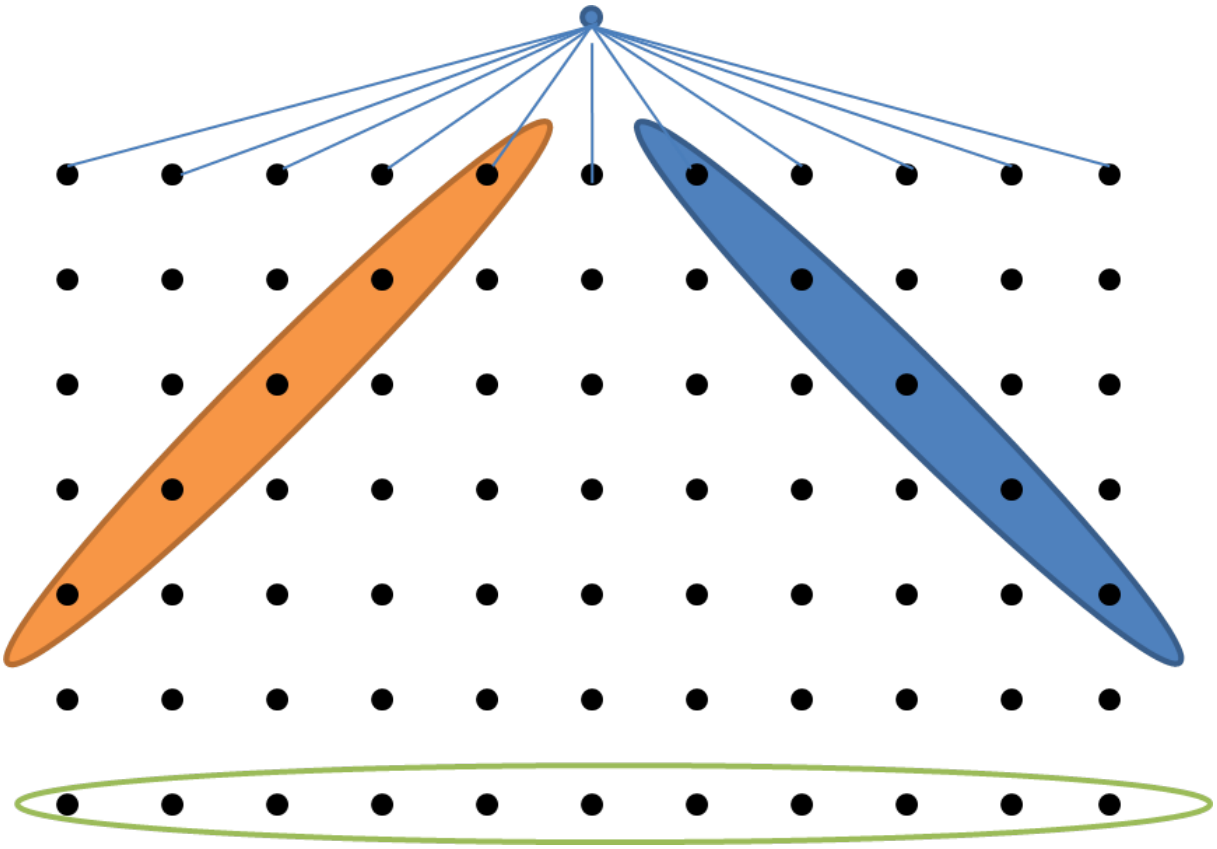

Figure 5.11: Explaining how Q_s can help search while still pushing diversity.

As a last point of discussion, we turn to the diversity of the solutions produced by *MQA*, MQA_D, and MQA_S. Figure C.4 shows the first solution of *MQA* on 30 problems of `BiSyn` . What is plotted here is a solution's final xCost (on the x-axis) and final yCost (on the y-axis). The columns of the plot indicate which metric was used to evaluate the quality as well as which queues were included in the search (as described in Table 5.3). The first and second columns of plots indicate the use of a single queue x or y that was also used to evaluate the final solution. The *MQA* variants successfully minimize either x or y alone. If x and y are collinear, as they are in *lin* and *nil*, then solutions tend to clump in the lower left corner. If x and y are negatively correlated, then solutions tend to be low in the minimized metric and high in the one that was ignored. More complex interactions lead to varying results.

The third, fourth and fifth columns of Figure C.4 all indicate solutions that are evaluated according to the z function, which is denoted by z being the first letter for all three

columns. However, each column indicates different kinds of queues that were used. The third column (z) indicates that only one queue, z, was included. It is clear that solutions which minimize both x and y are preferred. The fourth column (zxy) indicates that, while z was the evaluation metric used for the final plan quality, only the x and y queues were included. It is clear that employing separate queues for x and y not only led to a set of solutions that were distinct in both x and y but also tend to be distinct from those in the third column (z). This might suggest that putting all three queues together would lead to a better spread of solutions. However, we see that this is not the case in the fifth column ($zxyz$), which indicates that z is used to evaluate the final plan quality and three queues were used by search: x, y, and z. Adding the z metric to xy did not generally help with solution diversity but instead seems to have biased search toward minimizing the x axis; again this may be due to the sampling differences between x and y that were noted earlier at the start of Section 5.4.1 when discussing the impact of metric interaction.

Adding the diversity queue, Q_D, as shown for MQA_D in Figure C.6 does not seem to provide any significant gain, and when paired with nearly doubled runtime cost over MQA (cf. Figure 5.10) clearly indicates that diversity alone is not a good driver for generating diverse plan sets. Adding the parsimony queue, Q_S, as shown for MQA_S in Figure C.8 demonstrates exactly the kind of tradeoff we hoped to see. MQA_S generates a wider variety of single solutions for the cases where we evaluate using z (the third, fourth and fifth columns). It also generates diverse solutions across the gap that could not be found in either MQA and MQA_D. Finally, it is even able to generate more varied solutions for the collinear functions, which in MQA and MQA_D were clustered very near the origin. The drawback to this approach is that it generates this diversity at the expense of quality.

5.5 Results: Producing Plan Sets for `BiSyn`

We now turn our attention to extending the results for MQA on the `BiSyn` domain to producing plan sets. First, we examine the uniqueness of the plan sets for the `BiSyn` problem. Table 5.5 shows the results for obtaining 100 solutions for each function and metric combi-

Table 5.5: Solution counts and overlap for the `BiSyn` domain with MQA_T, MQA_{TD}, and MQA_{TS}.

				MQA_T		MQA_{TD}		MQA_{TS}	
	n	Avg	Total	Unique	Ratio	Unique	Ratio	Unique	Ratio
					x				
q	30	100.0	3000	2578	0.859	2288	0.763	2807	0.968
p	30	100.0	3000	2430	0.810	2208	0.736	2883	0.961
r	30	100.0	3000	2200	0.733	1984	0.661	2606	0.869
s	30	100.0	3000	2571	0.857	2197	0.732	2898	0.966
m	30	100.0	3000	2593	0.864	2251	0.750	2898	0.966
l	30	100.0	3000	1995	0.665	1502	0.501	2993	0.998
t	30	100.0	3000	2027	0.676	1677	0.559	2914	0.971
					y				
q	30	100.0	3000	2571	0.857	2156	0.719	2888	0.963
p	30	100.0	3000	2413	0.804	2283	0.761	2833	0.944
r	30	100.0	3000	2187	0.729	1938	0.646	2640	0.880
s	30	100.0	3000	2466	0.822	2234	0.745	2866	0.955
m	30	100.0	3000	2590	0.863	2175	0.725	2879	0.960
l	30	100.0	3000	2036	0.679	1558	0.519	2984	0.995
t	30	100.0	3000	2052	0.684	1638	0.546	2928	0.976
					z				
q	30	100.0	3000	2529	0.843	2187	0.729	2834	0.945
p	30	100.0	3000	2476	0.825	2180	0.727	2797	0.932
r	30	100.0	3000	2021	0.674	1644	0.548	2529	0.843
s	30	100.0	3000	2392	0.797	2186	0.729	2787	0.929
m	30	100.0	3000	2478	0.826	2174	0.725	2841	0.947
l	30	100.0	3000	1980	0.660	1354	0.451	2986	0.995
t	30	100.0	3000	1993	0.664	1341	0.447	2916	0.972
					xy				
q	30	100.0	3000	2795	0.932	2229	0.743	2905	0.968
p	30	100.0	3000	2630	0.877	2179	0.726	2858	0.953
r	30	100.0	3000	2452	0.817	2251	0.750	2699	0.900
s	30	100.0	3000	2759	0.920	2083	0.694	2878	0.959
m	30	100.0	3000	2822	0.941	2331	0.777	2902	0.967
l	30	100.0	3000	2232	0.744	1868	0.623	2796	0.932
t	30	100.0	3000	2444	0.815	2157	0.719	2924	0.975
					xyz				
q	30	100.0	3000	2705	0.902	2411	0.804	2875	0.958
p	30	100.0	3000	2500	0.833	2383	0.794	2814	0.938
r	30	100.0	3000	2361	0.787	2118	0.706	2641	0.880
s	30	100.0	3000	2762	0.921	2360	0.787	2844	0.948
m	30	100.0	3000	2737	0.912	2354	0.785	2888	0.963
l	30	100.0	3000	2105	0.702	1703	0.568	2650	0.883
t	30	100.0	3000	2101	0.700	1741	0.580	2556	0.852

nation of `BiSyn` . Each sub-table indicates the particular queue setup for these problems as detailed in Section 5.3.2 and Table 5.3. Every run was able to produce 100 solutions within the computational bounds, as indicated by the left three columns. The next three pairs of columns indicate the uniqueness ratio for each variant of MQA_T; we did not run without the tabu list because of the deliberate similarity of `BiSyn` and the security domain and our findings that *ITA* was best in the security domain (see Section 4.4.2). In comparing the number of unique solutions produced and the uniqueness ratios, where higher is better for both metrics, it is clear that a strict dominance relationship occurs: $MQA_{TD} \prec MQA_T \prec MQA_{TS}$. This is strong evidence that leveraging parsimony during search leads to improved search performance. However, these metrics do not provide evidence about how efficiently these plans are produced, the quality of those solutions, nor how well spread out the solutions across the metrics.

Figure C.9 shows the runtime distributions for MQA_T, MQA_{TD}, and MQA_{TS}. We can see that the search time per solution did not change much between these values and those of obtaining the first solution (cf. Figure 5.10). A Bonferroni adjusted t-test between the first solution and the 100 solutions reveals that many of the runtimes are not significantly different except for *lin* and *nil* for all three variants and *lin*, *nil*, and *lop* for MQA_{TD}.

In terms of quality, we can compare the solutions to the set of first solutions as well as between the three approaches that generated plan sets. Both comparisons reinforce a common thesis of this document: that solution quality and solution diversity are often at odds with each other. Figures C.10, C.12, and C.14 show the log-histograms for solution quality for obtaining the 100 plans in the problem, p00, of `BiSyn`; as before, these plots are placed on alternating pages to make comparison easier. When compared to the first solutions (cf. Figures C.3, C.5, and C.7, respectively), we see that solution quality has generally worsened, as shown by the higher prevalence of the larger right-most bars. This trend is less prevalent between the 1 and 100 solutions of MQA_{TS}, which suggests that the initial solution of MQA_{TS} already sacrifices quality. What remains to be seen is whether MQA_{TS} was able to continue the trend of a spread of solutions.

To understand the spread of the solutions, we examine Figures C.11 , C.13, and C.15. We see that the spread of solutions is most diverse for MQA_{TS}. Further, we see that MQA_{TS} was even able to produce diversity in the linear functions, where no other algorithm could do so.

5.6 Results: Producing Plan Sets for the Benchmarks

We close the results with a return to the benchmark domains from Chapter 4. Our goal in this section is to tie up two important themes that have run through this entire document by demonstrating two points. First, we show that driving search with parsimony is reasonable even for the benchmark domains because it yields a higher number of unique solutions and improved parsimony ratios over other approaches. Second, we want to show that selecting the right approach should depend on the needs of the application.

Table 5.6 compares four *MQA* variants with the original results from Table 4.4 for *ITA*. We can make two sets of comparisons in these results. First, when comparing the *MQA* variants, MQA_{TS} produces the highest number of unique solutions for all the domains except `Driverlog`, where MQA_D solved more in spite of achieving less coverage. This suggests that embedding a parsimony queue led to better results for uniqueness. However, we note that *ITA* was still able to achieve better uniqueness in the security domain for which it was designed. This suggests that selecting the right algorithm should depend on the needs of the application. Finally, Table 5.7 repeats the results for the parsimony ratio, s, from Table 4.3 as well as shows the result for s when using the *MQA*variants. We can see that using the parsimony queue leads to much higher values of s and that the minimums are usually higher. This suggests that using the parsimony queue naturally leads to more parsimonious solutions.

Table 5.8 compares the average diversity of MQA_{TS} with *ITA*, *LPGd*, and *RWS*. It is clear that (on average) MQA_{TS} generally produces more diverse plans than *ITA*, while rarely producing more diverse plans than *LPGd* and *RWS*. This is an expected result since *MQA* was shown to have more parsimonious plans than *LPGd* and *RWS*, so its average diversity will naturally be lower.

Table 5.6: Solution counts for the domains with the *MQA* variants and *ITA*(repeated from Table 4.4). The n column indicates the number of problems that were solved at least once for each domain by an algorithm.

	n	Avg	Total	Unique	Ratio
		MQA_D			
Depot	7	671.9	4703	133	0.028
DriverLog	11	661.2	7273	1908	0.262
Rover	20	402.6	8052	2210	0.274
Cybersec-strips	30	768.2	23047	843	0.037
Transport	20	969.0	19380	5743	0.296
		MQA_S			
Depot	9	830.1	7471	329	0.044
DriverLog	12	803.1	9637	482	0.050
Rover	20	927.2	18544	332	0.018
Cybersec-strips	28	968.9	27129	340	0.013
Transport	20	981.0	19619	196	0.010
		MQA_{TD}			
Depot	7	928.3	6498	8	0.001
DriverLog	12	798.1	9577	13	0.001
Rover	20	947.2	18944	21	0.001
Cybersec-strips	30	1000.0	30000	30	0.001
Transport	20	1000.0	20000	543	0.027
		MQA_{TS}			
Depot	8	336.0	2688	699	0.260
DriverLog	12	201.8	2422	391	0.161
Rover	20	519.2	10384	3713	0.358
Cybersec-strips	28	709.9	19876	1078	0.054
Transport	20	292.4	5849	3995	0.683
		ITA			
Depot	8	282.9	2263	246	0.109
DriverLog	13	205.2	2667	375	0.141
Rover	20	226.7	4533	314	0.069
Cybersec-strips	28	261.8	7331	1562	0.213
Transport	20	329.8	6595	5467	0.829

Table 5.7: The parsimony ratio from running the *MQA* variants on the benchmark problems.

RWS

	$\bar{x}$	SD	Med	Min
Depot	0.74	0.106	0.75	0.34
DriverLog	0.64	0.121	0.63	0.25
Rover	0.82	0.114	0.81	0.44
Cybersec	-	-	-	-
Transport	0.74	0.096	0.74	0.37

Div

	$\bar{x}$	SD	Med	Min
Depot	0.93	0.089	1.00	0.77
DriverLog	0.98	0.046	1.00	0.81
Rover	1.00	0.002	1.00	0.90
Cybersec	-	-	-	-
Transport	0.98	0.043	1.00	0.82

ITA

	$\bar{x}$	SD	Med	Min
Depot	1.00	0.018	1.00	0.72
DriverLog	0.99	0.040	1.00	0.61
Rover	1.00	0.011	1.00	0.83
Cybersec	-	-	-	-
Transport	0.95	0.070	1.00	0.50

Hybrid

	$\bar{x}$	SD	Med	Min
Depot	0.93	0.095	1.00	0.75
DriverLog	0.97	0.045	1.00	0.88
Rover	1.00	0.000	1.00	1.00
Cybersec	-	-	-	-
Transport	0.95	0.055	0.98	0.83

LPGd

	$\bar{x}$	SD	Med	Min
Depot	0.53	0.233	0.44	0.18
DriverLog	0.82	0.142	0.86	0.20
Rover	0.95	0.054	0.96	0.72
Cybersec	-	-	-	-
Transport	0.73	0.165	0.76	0.16

MQA_D

	$\bar{x}$	SD	Med	Min
Depot	0.98	0.111	1.00	0.27
DriverLog	0.91	0.149	1.00	0.20
Rover	0.98	0.039	1.00	0.70
cybersec	-	-	-	-
Transport	0.97	0.057	1.00	0.52

MQA_S

	$\bar{x}$	SD	Med	Min
Depot	0.97	0.049	1.00	0.63
DriverLog	0.98	0.048	1.00	0.72
Rover	0.99	0.021	1.00	0.83
cybersec	-	-	-	-
Transport	0.98	0.046	1.00	0.74

MQA_{TD}

	$\bar{x}$	SD	Med	Min
Depot	0.71	0.236	0.67	0.36
DriverLog	0.77	0.187	0.77	0.40
Rover	0.91	0.079	0.90	0.78
cybersec	-	-	-	-
Transport	0.90	0.083	0.90	0.63

MQA_{TS}

	$\bar{x}$	SD	Med	Min
Depot	0.96	0.055	1.00	0.48
DriverLog	0.99	0.042	1.00	0.57
Rover	1.00	0.016	1.00	0.77
cybersec	-	-	-	-
Transport	0.96	0.069	1.00	0.46

Table 5.8: Average stability diversity (D_s) comparison for the first 10 unique plans for *RWS*, *ITA*, MQA_{TS}, and *LPGd*. Each entry shows how often the column algorithm's average diversity for those ten solutions was better or equal to the row algorithm's average diversity ('$\geq$' columns), the union of problems with at least 10 solutions in either algorithm ('$\cup$' columns), and the average magnitude of difference ('$|\bar{\delta}|$' columns).

	ITA			*LPGd*			MQA_{TS}			*RWS*		
	$\geq$	$\cup$	$\lvert\bar{\delta}\rvert$	$\geq$	$\cup$	$\lvert\bar{\delta}\rvert$	$\geq$	$\cup$	$\lvert\bar{\delta}\rvert$	$\geq$	$\cup$	$\lvert\bar{\delta}\rvert$
Depot												
ITA		7		0	22	49.9	4	8	2.7	1	10	33.1
LPGd	22	22	49.9		22		22	22	45.4	20	22	22.5
MQA_{TS}	6	8	2.7	0	22	45.4		8		2	11	32.2
RWS	9	10	33.1	2	22	22.5	9	11	32.2		9	
Driverlog												
ITA		12		1	20	44.2	6	13	8.8	2	19	40.0
LPGd	19	20	44.2		20		17	20	12.1	10	20	28.8
MQA_{TS}	8	13	8.8	3	20	12.1		11		0	17	34.5
RWS	17	19	40.0	10	20	28.8	17	17	34.5		17	
Rover												
ITA		19		0	20	18.4	6	19	4.1	8	20	13.0
LPGd	20	20	18.4		20		20	20	15.1	16	20	3.3
MQA_{TS}	13	19	4.1	0	20	15.1		19		8	20	12.2
RWS	12	20	13.0	4	20	3.3	12	20	12.2		12	
Cybersec												
ITA		28	0	7	29	17.3	8	28	4.0	17	28	14.4
LPGd	22	29	17.3		24		23	29	17.1	18	25	16.7
MQA_{TS}	20	28	4.0	6	29	17.1		28		16	28	13.3
RWS	11	28	14.4	7	25	16.7	12	28	13.3		19	
Transport												
ITA		20		0	20	29.7	5	20	4.2	0	20	32.2
LPGd	20	20	29.7		20		20	20	26.5	9	20	11.1
MQA_{TS}	16	20	4.2	0	20	26.5		20		0	20	29.0
RWS	20	20	32.2	11	20	11.1	20	20	29.0		20	

5.7 Limitations

Our exploration of metric interaction has some limitations related to the domains we used, our algorithm approaches, and our evaluation methods.

One clear future direction is to enrich `BiSyn`. The current version does not have cycles or negative effects, which is justified by the security domain relying on the monotonicity property (see Motivation, Section 2.1.1 for details on monotonicity). But it is clear that other applications might introduce cycles (e.g. the production/consumption cycles or chains of actions leading to a (sub)goal as explored by Radzi [Rad11]), and many domains have actions with negative effects. The bicriteria functions we use (linear and curvilinear) are a good starting point, but they are simple and should be extended to more complex interactions and more than two metrics. We need to better control for the discretization of x that led to better results than y. Uniformly selecting x should solve this problem. Finally, there is some at least some evidence for plan-length correlation that should be explored in future work.

Our choice of MetricFF, A^*_ϵ and metric planning limits the findings to a single approach; we need to generalize our results to preference and cost-based planners as well as other metric planners. We expect that similar results will be found because many planners are still designed with the plan length and single-metric bias at their core. It is unclear how much the weight of A^*_ϵ or the heuristic accuracy could impact the search results, which needs to be addressed with a deeper study of the parameters for A^*_ϵ. It may also be fruitful to compare the plans found with worst cost plans rather than the optimal cost.

We are interested in combining this work with search for diverse alternatives plans (e.g., [Ngu+12; CMA11; Rob+12; Kho+13; Rad11; SL12]). We intend to generalize our set of domains to those by Radzi [Rad11] who examines the reliance of benchmarks on plan length and Sroka [SL12] who explores drives diversity by exploiting the metric sensitivity. We also hope to extend these findings to the security domain that motivates our research.

The variants of *MQA* only scratched the surface of what could be done with this algorithm, and the results presented for this algorithm are strictly empirical. One future direction

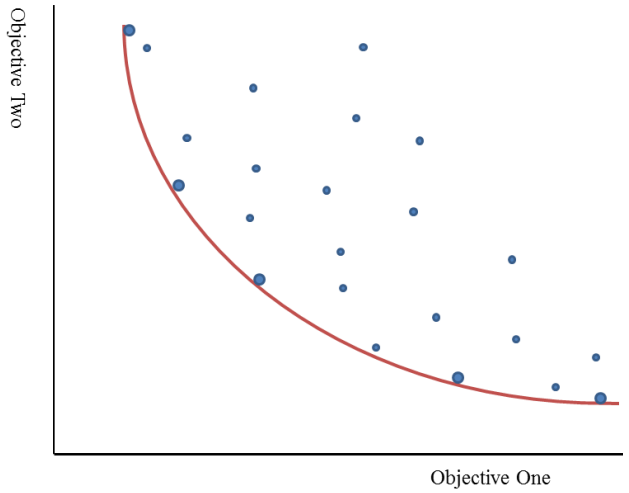

Figure 5.12: Example of a two-dimensional pareto front.

is to examine the potential of more complex queue selection strategies. Another direction is to characterize analytically the conditions under which *MQA* is optimal.

Finally, our evaluation of *MQA* provided anecdotal evidence that it produced solutions along a frontier of potential plans. A well understood way to evaluate solution diversity is to examine the coverage of solutions along a pareto-optimal front; Ehrgott provides a good introduction to the details of this field from an AI search perspective [Ehr08]. Figure 5.12 shows a two-dimensional pareto front. Feasible solutions to the problem are shown as dots. The large dots show a set of solutions that are said to be non-dominated with respect to at least one of the objectives. All other points not on the pareto front are dominated because a better solution exists for at least one objective. A clear future direction is to compare *MQA* to other approaches that can produce diverse solutions such as a population-based search algorithms like NSGA-II or the recent planning system DAE-YAHSP [Kho+13].

5.8 Revisiting the Tradeoffs of Generating Plan Sets

Our initial exploration into producing single solutions under bi-metric interaction yields several important insights. One of the more surprising findings is that A^*_ϵ search performs quite poorly when minimizing collinear functions ($y = x$ and $y = sigmoid(x)$), which suggests

that researchers should avoid combining x and y when they are (nearly) collinear. A^*_ϵ works well for curvilinear functions such as polynomials and an "inverted" sigmoid. Most critically, we discovered that scaling the metrics dramatically reduced search effectiveness. Poorer performance occurred when the metrics were uniformly scaled to control, at least in part, for plan-length correlation. Search performance also degraded as one metric was weighted more heavily.

The lack of a plan length for guidance in `BiSyn` is a serious hindrance to A^*_ϵ, which implicitly relies on relaxed heuristics for guiding search[13]. This justifies our approach of employing more than one queue to solve problems for this domain, where plan length was the same for all solutions. Using more than one queue to generate single solutions accomplishes at least two objectives. First, it allows us to vary systematically the bias of the algorithm. Second, it gives a solid foundation for extending the family of *MQA* approaches into producing plan sets.

We show that the *MQA* family solved many, but not all, of the problems associated with producing plan sets while simultaneously dealing with metric interaction. We create a parsimonious queue, Q_S, that first minimizes the reachability heuristic h_{relax} and then maximizes diversity. Combining Q_S with the Tabu mechanism of *ITA* results in more unique *and parsimonious* plans for many problems in `BiSyn` and the benchmarks. However, this is not always the case. For example, MQA_{TS} does *not* outperform *ITA* on the security domain for which *ITA* was originally designed.

These results both unify and clarify many of the themes we have observed. We combine the knowledge that quality and diversity interact with each other with the knowledge of how specific metric interactions impact search. Our findings led us to the key insight that parsimony – preferring shorter plans – is central to maximizing diversity while maintaining reasonable quality and efficiency. But we also show that no approach dominates. For exam-

[13]Thanks to Patrik Haslum for an insightful email discussion concerning this insight.

ple, *ITA* is still the best algorithm for `Cybersec` in generating the most unique solutions. Together these findings provide a strong case that the needs of the application should drive the choice of algorithm. It should also drive algorithm design. Perhaps most importantly, we have shown that analysis-directed changes to an algorithm can lead to improved performance and that the intrinsic design goal of plan length makes extensions difficult.

Chapter 6
Evaluating Diversity For Planners

This final chapter reviews the key findings so far from our exploration of the interplay of diversity, quality, and efficiency for search-based planning. Such systems are often evaluated under the assumption that they need to produce a single plan while minimizing a single quality metric, and a planner is preferred if it produces a better quality plan with the same (or fewer) computational resources. This quality-efficiency focus has naturally led to techniques that iteratively improve the next generation of planners to find that single best solution ever faster.

Our target application – a personalized security agent for home computer users – challenges the way in which many planning systems have been designed. The security application requires generating plan sets (as opposed to a single solution) and evaluating plans according to more than one quality metric (as opposed to a single metric). Figure 1.3 (left) demonstrates these two research directions, where the gray box indicates the current literature along both directions.

Moving in each of these two directions extends planner design and evaluation in two ways. The requirement of more than one metric expands the quality dimension. The requirement of producing plan sets introduces a new evaluation dimension of diversity. This creates the three axes of evaluation (the dependent variables) that we use in our study, as shown in Figure 1.3 (right). Many of the evaluation metrics are drawn from existing literature. For example, in the **efficiency metrics** we used CPU time and number of nodes generated. The **quality metrics** followed the common usage of plan cost for newer problems and plan length for older problems. Finally, we employ two **diversity metrics** from the literature that characterize the distance between plans within a plan set. One metric, $D_{\text{stability}}$, uses set difference between the plan sets, while the other, D_{norm}, normalizes the cardinality of the set intersection by the cardinality of the set union between two plans.

The existing diversity metrics have two shortcomings. None supports comparison *between* plan sets. So we create a metric, *overlap*, that takes the set intersection of two plan sets.

Within a plan set, plan distance does not characterize the distinctness of plans with respect to each other. We create two diversity metrics designed to alleviate this. We found in our studies that plans within a plan set could be subsets of each other. The first metric, *uniqueness*, captures the way in which plans do not subsume each other by removing any padded or permuted plans. The second metric, *parsimony ratio*, characterizes how verbose a planner is for a given plan π_l with respect to a minimal plan, π_k. We obtain π_k (for as many problems as possible) by running A* while restricting the allowed operators to only those in π_l.

Recent work has extended planning systems to produce diverse plan sets, and the dominant approach for driving search is to combine the quality and diversity metrics in a single, weighted objective function. While these approaches show they can improve the diversity of the final plan sets, relatively little is understood about the interplay of quality, efficiency, and diversity with respect to search behavior. Our goal is to examine this interplay with the hope of advancing the design of planning algorithms.

We tackle each research direction independently before combining them. Along the research direction of producing plan sets in a single objective, we assess several approaches for producing plan sets. Along the multiple metric direction, we examine metric interaction in a bicriteria synthetic problem. Along both research directions of producing plan sets within a bicriteria context, our work examines how to leverage the knowledge we gained. The culmination of our effort is the *MQA* algorithm that uses multiple queues to simultaneously manage separate quality and diversity metrics. *MQA* complements the existing algorithms by providing a way to keep metrics independent while also allowing search to leverage the bias of each one. In particular, it allows the simultaneous combination of maximizing diversity and minimizing the quality (or cost) metric(s). In the end, we show that understanding the interplay of diversity, quality, and efficiency while also considering the needs of the application can lead to an informed choice of which algorithm to use as well as lead to principled improvements to those algorithms.

6.1 Producing Plan Sets

As far as we are aware, our work is the first to evaluate several approaches for generating plan sets using the same experimental parameters and evaluation metrics. In total we use five approaches. Coman and Muñoz-Avila [CMA11] examine how to produce diverse plans in a weighted objective function for A*; our implementation is called *Div*. Nguyen et al. extend the LPG planner to produce diverse alternatives [Ngu+12] by adding a diversity component in the action selection heuristic; we call this extended planner *LPGd*. To these two existing approaches we add 1) a Random Walk Search (*RWS*) that randomly selects actions [XNM12], 2) an algorithm of our own creation called *ITA* [Rob+12] that uses a `(state, operator)` Tabu list to encourage exploration, and 3) a hybrid algorithm *Hybrid* that combines *Div* and *ITA* with the intent of leveraging their strengths.

We show that, *for the problems we studied*, combining the diversity metric with the quality metric(s) in a weighted objective function often leads to high diversity that sacrifices efficiency and quality. Our key contribution is showing that parsimony – preferring shorter plans while maximizing diversity – is essential to achieving diverse plan sets that maintain good quality. Our new diversity metric, the parsimony ratio, characterizes how well existing approaches find minimal-length plans that also maximize diversity. With this new diversity metric, we show that driving search with a weighted objective function including diversity leads to a greater proportion of plans that are not parsimonious, which has implications for multi-metric search in cases where we want to use diversity to drive search. In particular, we find a diversity-quality tradeoff in many domains, which leads us to explore metric interaction in a more principled way.

6.2 Understanding Metric Interaction

Since few current benchmark domains truly exercise the kinds of metric interaction we expect to see in the security domain, we create a synthetic domain. The synthetic domain, `BiSyn`, allows us to vary the interaction of two metrics, x and y, in a weighted objective

function, $z = x + y$, while fixing plan-length. These are both central requirements of our motivating security domain. We examine seven interactions: random (uncorrelated) interaction plus three kinds of metric interactions (and their mirrored counterparts): linear, sigmoidal, and polynomial.

We apply A^*_ϵ to `BiSyn` to understand the impact of the seven metric interactions. Instead of always selecting the top node from a queue, A^*_ϵ works by selecting from a focus list that is within a factor, ϵ, of the top solution. We observe the efficiency and quality of solutions produced by A^*_ϵ. We find that collinear interactions are challenging for A^*_ϵ, while curvilinear interactions are relatively easier. These findings have implications for any planner that employs a weighting scheme to manage its queues. We also found that scaling in either x or y while minimizing z reduces search effectiveness. This scaling issue has been observed by other researchers (e.g., [WR11; CBK11; SL12]) and has implications for how one manages differences in scale between metrics. Our method of controlling plan length while varying the interactions of two metrics gives another perspective from which to view the impact of cost-based search.

6.3 Unifying Parsimony and Metric Interaction: Multi-Queue A*

At long last, we arrive at our goal of generating plan sets in the context of multiple objectives. Here, the issues of the diversity-quality tradeoff, metric interaction, metric scale, and parsimonious diversity all conspire to thwart any approach that combines the metrics in a weighted objective function. Our solution is an iterated version of A* that we call Multi-Queue A* (MQA) that manages each quality metric, $t \in T$, in a separate queue, Q_t. These queues are minimized by using the usual (unweighted) A* function, $f = g + h$. To manage the diversity-quality tradeoff while still driving search toward parsimonious solutions, we create what we call a parsimony queue, Q_S, that minimizes first by the relaxed plan estimate, h_{relax}, then maximizes by the diversity, $D_{\text{stability}}$. We evaluate the performance of several variants of MQA on the `BiSyn` domain and find that the version enabling the parsimony queue (and the Tabu mechanism), MQA_{TS}, leads to an improved spread over the space of solutions for

only a modest loss of efficiency. However, it does so at the expense of plan quality, which is an expected result since we already know that diversity and quality can interact in subtle ways.

Our final study returns to the original benchmark domains of our study in an attempt to produce plan sets for a single quality metric. However, we have now arrived at the place we started and seen it for the first time. There are three important observations. First, we observe that diversity and quality often interact in a way that is collinear. Second, we have seen that the variants we studied of best first search are sensitive to metric interaction and it is worst when metrics interact in a collinear way. Third, we observe that search behavior is highly sensitive to scaling of the two metrics. We now understand why driving search with diversity on these problems leads to plans that violate parsimony, why solution quality drops, and why search does not produce better results with more effort. In particular, the design of many planners to minimize plan length (or to minimize quality, which is closely aligned with plan length) has led to some design decisions that work against producing good diversity. When we apply the *MQA* family to the benchmark domains, we find that a greater number of unique plans is produced and parsimony is improved for many benchmarks. As expected, solution quality is lower for these more diverse solutions. Finally, we also find that *ITA* still dominates for the security domain `Cybersec`, the domain for which it was specifically designed.

6.4 Limitations and Future Work

Naturally, there are a number of limitations to the work presented in this document. We will discuss a (surprising) hidden challenge that we faced before we move into the specific limitations motioned in Chapters 4 and 5 and how we see them leading into future work.

The Hidden Trade-offs in Generating Multi-metric Plan Sets

One of the most challenging aspects of this project was getting planners into the Mosaic framework that is discussed in Appendix A. In fact, we attempted to incorporate four differ-

ent planners into Mosaic over the past year and failed at each attempt. All attempts were failures because of the complexity of the software for these other planning systems. What is worse, each attempt cost about two weeks of effort before the failure manifested. It was a bit surprising given the relative ease with which we were able to build the initial Mosaic plugin for LAMA-2008. Nevertheless, it left us with little faith in trying to upgrade the LAMA planner to the latest version of Fast Downward (on which LAMA-2011 is based).

Classical planning is not itself difficult to understand: it is an implicit state transition system, usually encoded in textual (i.e., human-readable) form, that uses search to find solutions. In other words, any sophomore computer science or engineering student has the conceptual tools to understand how planning works – at least on the surface. Yet, the latest tutorial on *using* a planning system is over a decade old [Wel99], and that brings us to our hidden tradeoff. Planners are challenging to use, not just as black boxes, but as software systems. It *is* very challenging to tweak a planner without a great deal of effort. Trying to modify an existing planner in straightforward ways is surprisingly challenging as well. An argument could be made that the Fast Downward planner is easily accessible as a software project – especially the latest versions. However, the drawback of this planner is it requires a working understanding of the SAS$^+$ formalism, which is itself a step aside of the typical state-based planning system. We as a community need to change this if we want other (non-planning) software developers to embed planners into their systems.

One future direction for the Mosaic planner is to incrementally implement a non-SAS$^+$ planner such as MetricFF [Hof03]. Another is to provide a tutorial that helps ease the pain of embedding a planning system in another software project.

Algorithmic Approaches

Our use of algorithms instead of full planning systems was a double-edged sword. Using simple algorithms that are concise to describe and straightforward to implement allows us to be precise and confident in our observations. This is important because we found in previous work [RH09b] that the use of large planning systems confounded much of what we could say

at the end of a large study of 30 planners and over 5000 problems. However, the other side of the sword is that simple algorithms underperform (often by a large difference) when compared to state-of-the-art planning systems. This has the negative effect of limiting the available data for analysis. One goal for future work is to take a subset of specific enhancements that have provided the greatest benefit for planning systems and implement them in the algorithms we have used in this study. Future work should also explore alternative heuristics or metrics. Our current results then become a baseline for measuring which the enhancements improve performance.

Benchmark and Synthetic Domains

We selected four benchmark domains based on their use in previous work (i.e., `Depot`, `Driverlog`, and `Rover`) and based on similarity to our motivating security domain (i.e., `Cybersec`). We added another, `Transport`, to the set because it was a larger, newer domain that also made sense from the perspective of generating alternatives. These benchmarks are a subset of the available benchmarks, and we believe it would be worthwhile to assess the extent to which our findings relate to other domains. We generated 1000 solutions in the benchmark problems without actually verifying that these problems had this many solutions. It is unlikely that such a ceiling effect would have a big impact on our findings since we based most of our detailed analyses on the first $i = 10$ unique plans. Still this may have confounded some of the findings if the problem was too small for 1000 solutions and we need to verify (through some sort of exhaustive search) the extent to which this may have occurred. Another potential future direction is to examine the set of problems by Radzi [Rad11], who looked at how to generate problems with more interesting metrics.

The current version of the synthetic domain, `BiSyn`, was designed to mimic the security application while controlling for plan length and varying metric interactions in a known way. But many problems require other kinds of actions and effects and the interactions we studied were somewhat limited (and simple). Three directions for future work on `BiSyn` are to add more complex interactions, extend it to include cycles, and extend it to include delete actions.

The latter two directions would bring `BiSyn` closer to many of the existing benchmarks and allow us to say even more about how planning systems behave when metrics interact.

MQA

The use of multiple queues generated solutions for the bicriteria problem along a spectrum of minimizing x and minimizing y. Additionally, much more analysis can be done on *MQA*. It is not clear how close the solutions that were found fit on a pareto optimal front (i.e., they are non-dominated), so one future direction is to take a close look at the set of solutions from that perspective. Another promising area of future work is to possibly bound the runtime or solution quality using similar ideas from the bounded suboptimal search analysis.

6.5 Final Remarks

Like anything in life, what you measure is what improves. The evaluation metrics in efficiency and plan quality drove planner development for some 30 years. For example, when it was discovered that pairing state-based search with relaxed plan reachability heuristics [BLG97; McD99] dramatically improved search efficiency, planning with heuristic search became a dominant approach. Later, when the FF planning system [HN01] revealed that employing enforced hill climbing further improved performance, it became a favored approach. Finally, when the Fast Downward system showed employing a multivalued representation instead of propositional also led to a performance boost [Hel06], it became a dominant approach. The list goes on, to be sure.

It is often the case that the goals of efficiency and plan quality complement one another. For example, many planners use plan cost to drive search. But Radzi shows that, for the few metric benchmarks existing that have multiple metrics, these metrics rarely run counter to plan length [Rad11]. Despite the move to cost-based metrics, many of the cost-based benchmarks correlate strongly with plan length. Thus, planners that rely on plan length (i.e., reachability analysis in the form of the relaxed plan graph) will have an edge over those that do not. Given this perspective, it seems natural that the first foray into producing

plan sets would require only another weighted component in a planner's metric. After all, this is precisely what we (as a community) have learned from a long history of exploring the efficiency-quality tradeoff within (cost-based) planning. But our results suggest that this approach may not always be appropriate for the task at hand. We hope that our work provides convincing evidence:

- that combining diversity in a weighted metric has the unintended consequence that it often seriously misleads search with the undesirable tradeoffs of decreased efficiency and poor quality;
- that the weighted linear combination of multiple metrics confounds searching for good quality plans that *balance* the objectives at hand precisely when the metrics interact in unexpected ways or when the metrics are pathologically intertwined (i.e., they fall in the same range but are contradictive);
- that, for planning, diversity is at odds with quality because long plans usually imply poor quality, thus further confounding the weighted objective function;
- that the two directions of multiple metrics and plan sets are an under-explored area of planning worthy of further work;
- that we have simple, direct ways to harness algorithm (or metric, or heuristic) bias to drive search in appropriate directions and that the portfolio community can teach us a great deal about how to do this; and, finally,
- that diverse plans are only worthwhile if they are also parsimonious.

If what we measure is indeed what improves, then let us take stock, regather, and move forward with a refined vision of how we will evaluate the next generation of classical planning systems.

Bibliography

[AIP98] AIPS-98 Int'l Planning Competition Committee. *PDDL : the planning domain defnition language*. Tech. rep. The AIPS-98 committee consisted of: M. Ghallab and A. Howe and C. Knoblock and D. McDermott and A. Ram and M. Veloso and D. Weld and D. Wilkins. Yale Center for Computational Vision and Control, 1998.

[AWK02] P. Ammann, D. Wijesekera, and S. Kaushik. "Scalable, Graph-Based Network Vulnerability Analysis". In: *Proc. of the 9th ACM Conf. on Computer and Communications Security*. Washington, DC, USA, 2002, pp. 217–224.

[Bac01] F. Bacchus. "AIPS'00 Planning Competition". In: *AI Magazine* 22(3) (2001), pp. 47–56.

[Bar+03] L. Barbulescu, J. Watson, L. Whitley, and A. Howe. "Scheduling Space-Ground Communications for the Air Force Satellite Control Network". In: *Journal of Scheduling* 7.1 (2003), pp. 7–34.

[BDK09] J. Benton, M. B. Do, and S. Kambhampati. "Anytime Heuristic Search for Partial Satisfaction Planning". In: *Artificial Intelligence* 173.5-6 (2009), pp. 562–592.

[Ber+08] P. Berry, B. Bulka, B. Peintner, M. Roberts, and N. Yorke-Smith. "Neptune: A Mixed-Initiative Environment for Planning and Scheduling". In: *Proc. of the Twenty-First Int'l Florida Artificial Intelligence Research Society Conference (FLAIRS-08)*. Coconut Grove, FL, 2008, pp. 573–574.

[BF95] A. Blum and M. Furst. "Fast Planning Through Planning Graph Analysis". In: *Proc. of the 14th Int'l Joint Conf. on Artificial Intelligence (IJCAI-95)*. (Montreal, Quebec, Canada). 1995, pp. 1636–1642.

[BG01] B. Bonet and H. Geffner. "Planning as heuristic search". In: *Artificial Intelligence* 129.1-2 (2001), pp. 5–33.

[BLG97] B. Bonet, G. Loerincs, and H. Geffner. "A Robust and Fast Action Selection Mechanism for Planning". In: *Proc. 14th National Conf. on Artificial Intelligence*. (Providence, Rhode Island, USA). 1997.

[BN95] C. Bäckström and B. Nebel. "Complexity Results for SAS+ Planning". In: *Computational Intelligence* 11.4 (1995), pp. 625–655.

[Bod+] M. Boddy, J. Gohde, J. T. Haigh, and S. Harp. "Course of Action Generation for Cyber Security Using Classical Planning". In: *Proc. of the 15th Int'l Conf. on Automated Planning and Scheduling (ICAPS-05)*. (Monterey, CA, USA). Menlo Park, CA: AAAI Press, pp. 12–21.

[BS00] M. Becker and S. Smith. "Mixed-Initiative Resource Management: The AMC Barrel Allocator". In: *Proc. of the 5th Int'l Conf. on AI Planning and Scheduling*. 2000, pp. 32–41.

[Byr+12] Z. Byrne et al. "Perceptions of Internet Threats: Behavioral Intent to Click Again". In: *Proc. of the 27th Annual Society for Industrial and Organizational Psychology (SIOP) Conf.* San Diego, CA, USA, 2012.

[CBK11] W. Cushing, J. Benton, and S. Kambhampati. "Cost Based Satisficing Search Considered Harmful". In: *Working notes of the Workshop on Heuristics for Domain-independent Planning at Proc. of the 21st Int'l Conf. on Automated Planning and Scheduling (ICAPS-11)*. (Freiburg, Germany, June 11-16). Menlo Park, CA: AAAI Press, 2011.

[Chi+00] S. Chien et al. "ASPEN - Automating Space Mission Operations using Automated Planning and Scheduling". In: *6th Int'l SpaceOps Symposium (Space Operations)*. Toulouse, France, 2000.

[CMA11] A. Coman and H. Muñoz-Avila. "Generating Diverse Plans Using Quantitative and Qualitative Plan Distance Metrics". In: *Proc. of the 25th AAAI Conf. on Artificial Intelligence (AAAI-11)*. (San Francisco, CA, USA. Aug. 7-11). Menlo Park, CA.: AAAI Press, 2011, pp. 946–951.

[Coh+89] P. Cohen, M. Greenberg, D. M. Hart, and A. E. Howe. "Trial By Fire: Understanding the Design Requirements for Agents in Complex Environments". In: *AI Magazine* 10.3 (1989), pp. 32–48.

[Col+12] A. J. Coles, A. I. Coles, M. Fox, and D. Long. "COLIN: Planning with Continuous Linear Numeric Change". In: *Journal of Artificial Intelligence Research* 44 (2012), pp. 1–96.

[CWH06] Y. Chen, B. W. Wah, and C.-W. Hsu. "Temporal Planning using Subgoal Partitioning and Resolution in SGPlan". In: *Journal of Artificial Intelligence Research* 26 (2006), pp. 323–369.

[Dew+07] R. Dewri, N. Poolsappasit, I. Ray, and D. Whitley. "Optimal security hardening using multi-objective optimization on attack tree models of networks". In: *Proc. of the 14th ACM conference on Computer and communications security (CCS-07). Alexandria, Virginia, USA*. Alexandria, Virginia, USA: ACM, 2007, pp. 204–213.

[DK03] M. B. Do and S. Kambhampati. "SAPA: A Multi-objective Metric Temporal Planner." In: *Journal of Artificial Intelligence Research* 20 (2003), pp. 155–194.

[DK04] M. B. Do and S. Kambhampati. "Partial Satisfaction (Over-Subscription) Planning as Heuristic Search". In: *Proc. of Knowledge Based Computer Systems (KBCS-2004)*. 2004.

[EHN94a] K. Erol, J. Hendler, and D. S. Nau. "HTN Planning: Complexity and Expressivity". In: *Proc. of the 12th National Conf. on Artificial Intelligence (AAAI-94)*. Menlo Park, California: AAAI Press, 1994, pp. 1123–1128.

[EHN94b] K. Erol, J. Hendler, and D. S. Nau. *Semantics for HTN Planning*. Tech. rep. CS-TR-3239. Also cross-referenced as ISR-TR-95-9, Also cross-referenced as UMIACS-TR-94-31. University of Maryland, 1994.

[EHN94c] K. Erol, J. Hendler, and D. S. Nau. "UMCP: A Sound and Complete Procedure for Hierarchical Task-Network Planning". In: *Proc. of the 2nd Int'l Conf. on Artificial Intelligence Planning Systems (AIPS-94)*. Ed. by K. J. Hammond. Menlo Park, California: AAAI Press, 1994.

[Ehr08] M. Ehrgott. "Multiobjective Optimization". In: *AI Magazine* 29 (2008), pp. 47–57.

[Ero+95] K. Erol, J. Hendler, D. S. Nau, and R. Tsuneto. "A Critical Look at Critics in HTN Planning". In: *Proc. of the 14th Int'l Joint Conf. on Artificial Intelligence (IJCAI-95)*. (Montreal, Quebec, Canada). 1995.

[FJ03] J. Frank and A. Jonsson. "Constraint-based Attribute and Interval Planning". In: *Journal of Constraints* 8.4 (2003), 339–364.

[FL03] M. Fox and D. Long. "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains". In: *Journal of Artificial Intelligence Research* 20 (2003), pp. 61–124.

[FL06] M. Fox and D. Long. "Modelling Mixed Discrete-Continuous Domains for Planning". In: *Journal of Artificial Intelligence Research* 27 (2006), pp. 235–297.

[FN71] R. E. Fikes and N. J. Nilsson. "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving". In: *Artificial Intelligence* 2.3-4 (1971), pp. 189–208.

[Fow+05] J. W. Fowler, B. Kim, W. M. Carlyle, E. S. Gel, and S.-M. Horng. "Evaluating solution sets of 'a posteriori' solution techniques for bi-criteria combinatorial optimization problems". In: *Journal of Scheduling* 8 (1 2005), pp. 75–96.

[Fox+06] M. Fox, A. Gerevini, D. Long, and I. Serina. "Plan Stability: Replanning versus Plan Repair". In: *Proc. of the 16th Int'l Conf. on Automated Planning and Scheduling (ICAPS-06)*. (Lake District, Cumbria, UK, June 6-10). Menlo Park, CA: AAAI Press, 2006, 212–221.

[Fox94] M. Fox. "Intelligent Scheduling". In: ed. by M. Zweben and M. Fox. San Francisco, CA: Morgan Kaufmann Pub. Co., 1994. Chap. ISIS: A Retrospective, pp. 3–28.

[Fue11] R. Fuentetaja. "The CBP planner". In: *The 2011 International Planning Competition: Description of Participating Planners, Deterministic Track*. Ed. by A. Garcia-Olaya, S. Jimenez, and C. L. Lopez. 2011, pp. 21–24.

[Ger+09] A. E. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos. "Deterministic planning in the fifth int'l planning competition: PDDL3 and experimental evaluation of the planners". In: *Artificial Intelligence* 173.56 (2009). Advances in Automated Plan Generation, pp. 619 –668.

[GG12] N. Ghosh and S. Ghosh. "A Planner-based Approach to Generate and Analyze Minimal Attack Graph". In: *Int'l Journal of Applied Intelligence* 36.2 (2012). Published online: 25 November 2010, pp. 369–390.

[GL05] A. Gerevini and D. Long. *Plan Constraints and Preferences in PDDL3*. Tech. rep. RT 2005-08-47. Dept. of Electronics for Automation, University of Brescia, Italy, 2005.

[GNT04] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning*. Morgan Kaufmann Publishers, 2004.

[Gol+02] R. Goldman, K. Haigh, D. Musliner, and M. Pelican. "MACBETH: a multi-agent constraint-based planner [autonomous agent tactical planner]". In: *Proc. of the 21st Digital Avionics Systems Conference*. Vol. 2. 2002, pp. 31–38.

[GS02] A. Gerevini and I. Serina. "LPG: a Planner based on Local Search for Planning Graphs". In: *Proc. of the 6th Int'l Conf. on Artificial Intelligence Planning Systems (AIPS-02)*. (Toulouse, France. April 23-27). Ed. by M. Ghallab, J. Hertzberg, and P. Traverso. Menlo Park, CA: AAAI Press, 2002, pp. 13–22.

[GSS03] A. Gerevini, A. Saetti, and I. Serina. "Planning Through Stochastic Local Search and Temporal Action Graphs in LPG". In: *Journal of Artificial Intelligence Research* 20 (2003), pp. 239–290.

[GSS06] A. Gerevini, A. Saetti, and I. Serina. "An Approach to Temporal Planning and Scheduling in Domains with Predicatable Exogenous Events". In: *Journal of Artificial Intelligence Research* 25 (2006), pp. 187–231.

[HBG05] P. Haslum, B. Bonet, and H. Geffner. "New Admissible Heuristics for Domain-Independent Planning". In: *Proc. of the 20th National Conf. on Artificial Intelligence (AAAI-05)*. (Pittsburgh, Pennsylvania, USA, July 9-13, 2005). Menlo Park, CA: AAAI Press, 2005, pp. 1163–1168.

[HE05] J. Hoffmann and S. Edelkamp. "The Deterministic Part of IPC-4: An Overview". In: *Journal of Artificial Intelligence Research* 24 (2005), pp. 519–579.

[Hel06] M. Helmert. "The Fast Downward Planning System". In: *Journal of Artifical Intelligence Research* 26 (2006), pp. 191–246.

[HG00] P. Haslum and H. Geffner. "Admissible Heuristics for Optimal Planning". In: *Proc. of the 5th Int'l Conf. on Artificial Intelligence Planning Systems (AIPS-00)*. (Breckenridge, CO, USA). Menlo Park, CA: AAAI Press, 2000, pp. 140–149.

[HL03] R. Howey and D. Long. "VAL's Progress: The Automatic Validation Tool for PDDL2.1 used in the International Planning Competition". In: *Workshop on The Competition: Impact, Organization, Evaluation, Benchmarks, Working Notes of the Int'l Conf. on Automated Planning and Scheduling*. (Trento, Italy, June). 2003.

[HN01] J. Hoffmann and B. Nebel. "The FF Planning System: Fast Plan Generation Through Heuristic Search". In: *Journal of Artificial Intelligence Research* 14 (2001), pp. 253–302.

[Hof03] J. Hoffmann. "The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables". In: *Journal of Artificial Intelligence Research* 20 (2003), pp. 291–341.

[How93] A. E. Howe. *Evaluating Planning Through Simulation: An Example Using Phoenix*. Spring Symposiium SS-93-03. Menlo Park, California: AAAI, 1993.

[Key10] E. Keyder. "New Heuristics For Planning With Action Costs". PhD thesis. Department of Information and Communication Technologies, Universitat Pompeu Fabra, 2010.

[KG08] E. Keyder and H. Geffner. "Heuristics for Planning with Action Costs Revisited". In: *Proc. of the European Conf. in AI (ECAI-08)*. (Amsterdam, Netherlands). Vol. 178. 2008, pp. 588–592.

[KG09] E. Keyder and H. Geffner. "Soft Goals Can Be Compiled Away". In: *Journal of Artificial Intelligence Research* 36 (2009), pp. 547–556.

[Kho+13] M.-R. Khouadjia, M. Schoenauer, V. Vidal, J. Dréo, and P. Savéant. "Multi-Objective AI Planning: Evaluating DAEyahsp on a Tunable Benchmark". In: *Proc. of the 7th International Conference on Evolutionary Multi-Criterion Optimization (EMO-2013)*. (Sheffield UK, March 19-22). LNCS 7811. Sheffield, UK: Springer, Mar. 2013, pp. 36–50.

[KLC98] L. Kaelbling, M. Littman, and A. Cassandra. "Planning and acting in partially observable stochastic domains." In: *Artificial Intelligence* 101 (1998), pp. 99–134.

[KLM96] L. P. Kaelbling, M. L. Littman, and A. W. Moore. "Reinforcement Learning: A Survey". In: *Journal of Artificial Intelligence Research* 4 (1996), pp. 237–285.

[LF03] D. Long and M. Fox. "The 3rd Int'l Planning Competition: Results and Analysis". In: *Journal of Artificial Intelligence Research* 20 (2003), pp. 1–59.

[McD99] D. McDermott. "Using Regression-Match Graphs to Control Search in Planning". In: *Artificial Intelligence* 109.1-2 (1999), pp. 111–159.

[MEL01] A. Moore, R. Ellison, and R. Linger. *Attack Modeling for Information Security and Survivability*. Tech. rep. CMU/SEI-2001-TN-001. Software Engineering Institute, Carnegie Melon University, 2001.

[Mye+02] K. L. Myers, W. M. Tyson, M. J. Wolverton, P. A. Jarvis, T. J. Lee, and M. desJardins. "PASSAT: A User-centric Planning Framework". In: *Proc. of the 3rd Int'l NASA Workshop on Planning and Scheduling for Space*. (Houston, TX, USA). 2002.

[Nau+03] D. S. Nau et al. "SHOP2: An HTN Planning System". In: *Journal of Artificial Intelligence Research* 20 (2003), pp. 379–404.

[Ngu+12] T. Nguyen, M. Do, A. Gerevini, I. Serina, B. Srivastava, and S. Kambhampati. "Generating Diverse Plans to Handle Unknown and Partially Known User Preferences". In: *Artificial Intelligence* 190 (2012), pp. 1–31.

[OSR10] J. L. Obes, C. Sarraute, and G. Richarte. "Attack Planning in the Real World". In: *Workshop on Security and Artificial Intelligence (SecArt-10), Working Notes of the 24th AAAI Conf. on Artificial Intelligence (AAAI-10)*. Atlanta, USA, 2010.

[PK82] J. Pearl and J. H. Kim. "Studies in Semi-Admissible Heuristics". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 4.4 (1982), pp. 392–399.

[Pol02] M. E. Pollack. "Planning technology for intelligent cognitive orthotics." In: *Proc. of the 12th Int'l Conf. on Automated Planning and Scheduling (ICAPS-02)*. (Toulouse, France). Menlo Park, CA: AAAI Press, 2002.

[Pol+03] M. E. Pollack et al. "Autominder: An Intelligent Cognitive Orthotic System for People with Memory Impairment". In: *Robotics and Autonomous Systems* 44.3-4 (2003), pp. 273–282.

[PS98] C. Phillips and L. Swiler. "A Graph-Based System for Network-Vulnerability Analysis". In: *Proc. of the New Security Paradigms Workshop*. Chicago, IL, 1998, pp. 71–79.

[Rad11] M. Radzi. "Multi-objective planning using linear programming". PhD thesis. The University of Strathclyde, 2011.

[Rap] Rapid7. *Nexpose Community Edition*. Available from http://www.rapid7.com/products/nexpose-community-edition.jsp.

[RH09a] S. Richter and M. Helmert. "Preferred Operators and Deferred Evaluation in Satisficing Planning". In: *Proc. of the 19th Int'l Conf. on Automated Planning and Scheduling (ICAPS-09)*. (Thessaloniki, Greece, September 19-23). Menlo Park, California: AAAI Press, 2009, pp. 273–280.

[RH09b] M. Roberts and A. Howe. "Learning from Planner Performance". In: *Artificial Intelligence* 173 (2009), pp. 536–561.

[RHF07] M. Roberts, A. Howe, and L. Flom. "Learned Models of Performance for Many Planners". In: *Workshop on Learning and Planning, Working Notes of the 17th Int'l Conf. on Automated Planning and Scheduling (ICAPS-07)*. (Providence, Rhode Island, USA. September 22-26). Menlo Park, California: AAAI Press, 2007.

[Rob+11] M. Roberts, A. Howe, I. Ray, M. Urbanska, Z. S. Byrne, and J. M. Weidert. "Personalized Vulnerability Analysis through Automated Planning". In: *Workshop on Security and Artificial Intelligence (SecArt-11), Working Notes of the 22nd Int'l Joint Conf. on Artificial Intelligence (IJCAI-11)*. (Barcelona, Catalonia, Spain. July 16-22). 2011.

[Rob+12] M. Roberts, A. E. Howe, I. Ray, and M. Urbanska. "Using Planning for a Personalized Security Agent". In: *Workshop on Problem Solving using Classical Planners, Working Notes of the 26th AAAI Conf. on Artificial Intelligence (AAAI-12)*. (Toronto, Ontario, Canada. July 22-26). Menlo Park, CA.: AAAI Press, 2012.

[RW10] S. Richter and M. Westphal. "The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks". In: *Journal of Artificial Intelligence Research* 39 (2010), pp. 127–177.

[SBK04] S. Smith, M. Becker, and L. Kramer. "Continuous Management of Airlift and Tanker Resources: A Constraint-Based Approach". In: *Mathematical and Computer Modeling – Special Issue on Defense Transportation: Algorithms, Models and Applications for the 21st Century* 39.6-8 (2004), pp. 581–598.

[Sch+03] D. Schreckenghost, M. B. Hudson, C. Thronesbery, and K. Kusy. "When Automated Planning is Not Enough: Assisting Users in Building Human Activity Plans". In: *Proceeding of the 7th Int'l Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS-2003)*. (NARA, Japan). 2003.

[Sch+09] P. Schermerhorn, J. Benton, M. Scheutz, K. Talamadupula, and S. Kambhampati. "Finding and Exploiting Goal Opportunities in Real-time during Plan Execution". In: *Proc. of the Int'l Conf. on Intelligent Robots and Systems (IROS)*. 2009, pp. 3912–3917.

[SEM02] B. D. Smith, B. E. Engelhardt, and D. H. Mutz. "The RADARSAT-MAMM Automated Mission Planner". In: *AI Magazine* 23 (2002), pp. 25–36.

[SFJ00] D. Smith, J. Frank, and A. Jonsson. "Bridging the gap between planning and scheduling". In: *Knowledge Engineering Review* 15.1 (2000), pp. 61–94.

[She+02] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing. "Automated Generation and Analysis of Attack Graphs". In: *Proc. of the IEEE Symposium on Security and Privacy*. Oakland, CA, USA, 2002, pp. 273–284.

[She+98] R. Sherwood, A. Govindjee, D. Yan, G. Rabideau, S. Chien, and A. Fukunaga. "Using ASPEN to Automate EO-1 Activity Planning". In: *Proc. of the 1998 IEEE Aerospace Conf.* Aspen, CO, 1998, pp. 145–152.

[SL12] M. Sroka and D. Long. "Exploring Metric Sensitivity of Planners for Generation of Pareto Frontiers". In: *Starting AI Researchers (STAIRS 2012)*. Vol. 241. Frontiers in Artificial Intelligence and Applications. 2012, pp. 306–317.

[SLK02] S. Scott, N. Lesh, and G. Klau. "Investigating Human-Computer Optimization." In: *Proc. of Conf. on Human Factors in Computer Systems (CHI-2002)*. Minneapolis, MN, 2002, pp. 155–162.

[Sri+07] B. Srivastava, S. Kambhampati, T. Nguyen, M. Do, A. Gerevini, and I. Serina. "Domain Independent Approaches for Finding Diverse Plans". In: *Proc. of the 20th Int'l Joint Conf. on Artificial Intelligence*. 2007, pp. 2016–2022.

[Sub97] K. Subbarao. "Refinement Planning as a Unifying Framework for Plan Synthesis". In: *AI Magazine* 18(2) (1997), pp. 67–97.

[Tal+10] K. Talamadupula, J. Benton, S. Kambhampati, P. Schermerhorn, and M. Scheutz. "Planning for Human-Robot Teaming in Open Worlds". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 1.2 (2010), pp. 1–24.

[Urb+13] M. Urbanska, M. Roberts, I. Ray, A. Howe, and Z. Byrne. "Accepting the Inevitable: Factoring the User into Home Computer Security". In: *3rd ACM Conf. on Data and Application Security and Privacy (CODASPY-13)*. (San Antonio, TX). 2013.

[Vid04] V. Vidal. “A Lookahead Strategy for Heuristic Search Planning”. In: *Proc. of the 14th Int’l Conf. on Automated Planning and Scheduling (ICAPS-04)*. (Whistler, BC, Canada). Menlo Park, CA: AAAI Press, 2004, pp. 150–159.

[Wel94] D. S. Weld. “An Introduction to Least Commitment Planning”. In: *AI Magazine* 15.4 (1994), pp. 27–61.

[Wel99] D. S. Weld. “Recent Advances in AI Planning”. In: *AI Magazine* 20.2 (1999), pp. 93–123.

[WLB03] D. E. Wilkins, T. J. Lee, and P. Berry. “Interactive Execution Monitoring of Agent Teams”. In: *Journal of Artificial Intelligence Research* 18 (2003), pp. 217–261.

[WR11] C. Wilt and W. Ruml. “Cost-Based Heuristic Search is Sensitive to the Ratio of Operator Costs”. In: *Proc. of the 4th Annual Symposium on Combinatorial Search (SoCS)*. (Barcelona, Spain). AAAI Press. Menlo Park, CA, 2011, pp. 172–179.

[XNM12] F. Xie, H. Nakhost, and M. Müller. “Planning via Random Walk-Driven Local Search.” In: *Proc. of the 22nd Int’l Conf. on Automated Planning and Scheduling (ICAPS-12)*. (Atibaia, Sao Paulo, Brazil, June 25-29). Palo Alto, California: AAAI Press, 2012, pp. 315–322.

Appendices

Appendix A
The Mosaic Planning Framework

The planning framework we use in this study, called Mosaic, is designed to allow planners to be plugged into it while the algorithms, memory management, logging and experimental instrumenting facilities remain constant. It was designed to allow a fair comparison between distinct planner implementations. Since most planners are implemented in C++ , Mosaic is also implemented in C++ . Mosaic configures itself based on XML files passed in at runtime that allow the user to modify the heuristics, queues, and search strategies used for search.

Figure A.1 shows the interfaces of the main components. Each interface provides a consistent way to access concrete subclasses. At the center of the diagram is the `Planner` and `SearchManager` that sets up the various components based on XML configuration files. The PlannerManager first instantiates the underlying planner as a NeighborhoodManager (lower right). The NeighborhoodManager allows the successor generator of the planner to be wrapped in a consistent interface. The `StateFactory` manages memory for Mosaic.

The PlannerManager constructs the appropriate heuristics for the current configuration (middle right). Heuristics can be compound and are used both to sort queues (upper left) and evaluate states; it is a slight misnomer to call some sorting mechanisms "heuristics" but it serves the overall structure of the planner, since components understand how to work with the IHeuristic interface. For example, the F-heuristic used to sort the queue for A* is composed of the DistanceToGoHeuristic and the h-heuristic. A heuristic's value can be (optionally) stored in a `SearchNodeWithHeuristic` to be used by other components.

A `SearchStrategy` is used by the planner to search for a plan. The `step()` function of this class is distinct for each strategy (e.g., A*, Random Walk Search, Iterated WA*). Options for each strategy can be passed into the planner via the XML files. Each Strategy uses one or more Queues to manage its state as search progresses. Queues are themselves sorted according to a Comparator functor object. For example, a queue for A* simply sorts the Queue according to $f = g + h$, while the parsimony queue, Q_S, (see Section 5.3.2) minimizes h before maximizing diversity.

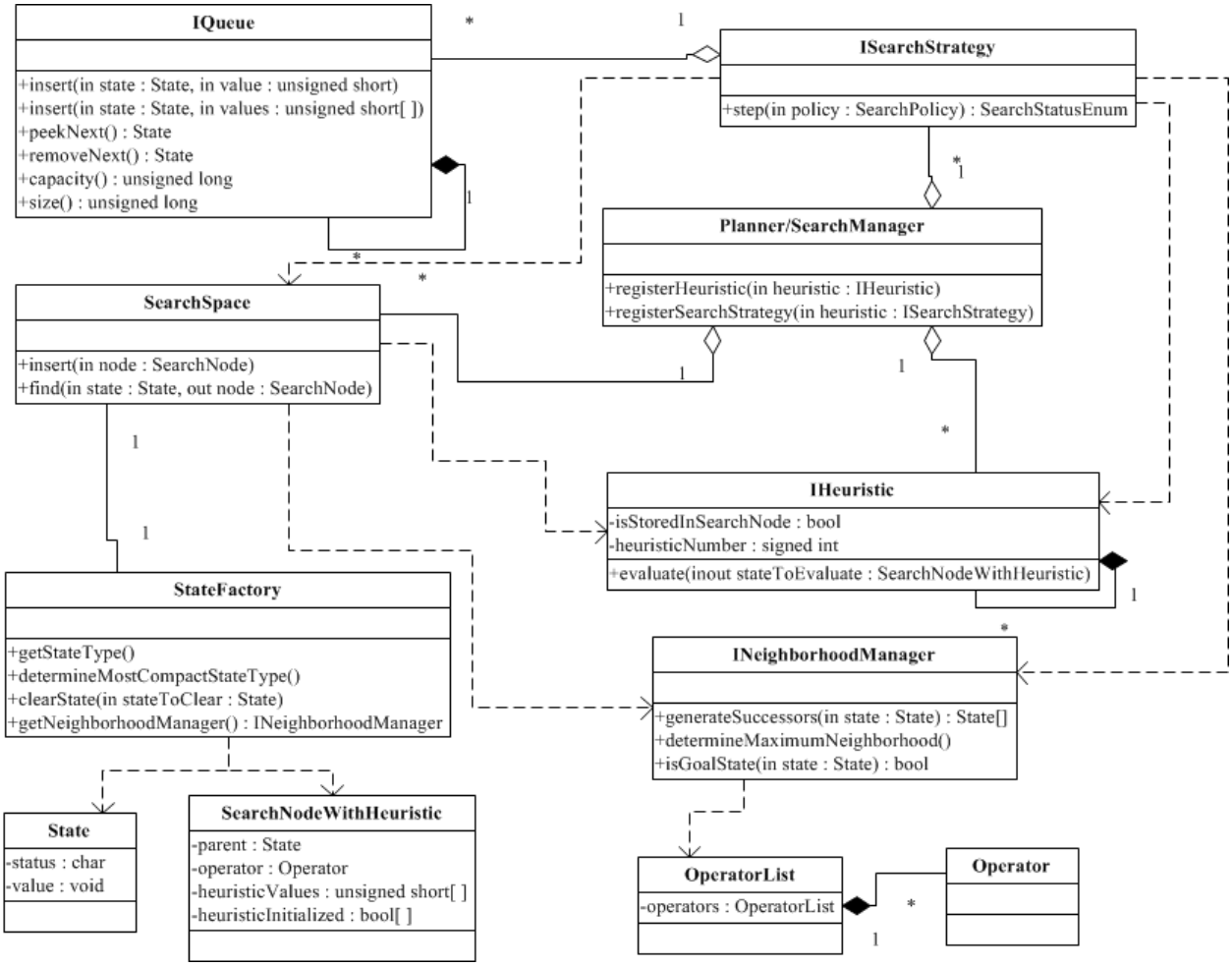

Figure A.1: The main components of the Mosaic Planner Architecture.

Appendix B
Supplemental Plots for Chapter 4

```
(define (domain attack-graph)
(:requirements :strips :equality :disjunctive-preconditions :typing )
(:types
  Object
  Action
  ExploitState
  software
)
(:predicates
  (action-observed ?Action - Action)
  (action-taken ?Action - Action)
  (noop-predicate ?Noop - Object)
  (exploit-occurred  ?Exploit - ExploitState)
  (software-installed  ?Software - software) )

(:action Exploit_DenialOfService_1
  :parameters (
      ?Action2 - Action ;ObeservedState_CVE_2010_0187_Exploited_2
      ?Action8 - Action ;ObeservedState_CVE_2008_3111_SunJavaMultiple_Exploited_8
      ?Action14 - Action ;ObeservedState_CVE_2010_4091_OS_Exploited_14 )
  :precondition
    (or
      (and
        (action-taken ?Action2)
        (= ?Action2 ObeservedState_CVE_2010_0187_Exploited_2) )
      (and
        (action-taken ?Action8)
        (= ?Action8 ObeservedState_CVE_2008_3111_SunJavaMultiple_Exploited_8) )
      (and
        (action-taken ?Action14)
        (= ?Action14 ObeservedState_CVE_2010_4091_OS_Exploited_14) ) )
  :effect (and (exploit-occurred Exploit_DenialOfService_1) ) )

(:action ObeservedState_CVE_2010_0187_Exploited_2
  :parameters (
      ?Dependency3 - software
      ?Action5 - Action ;AttackAction_FlashFileCompromised_5
      ?Action6 - Action ;UserAction_UserOpensFlashFile_6 )
  :precondition
    (and
        (software-installed ?Dependency3)
        (= ?Dependency3 Adobe_Flash_6_0_88_0)
        (action-taken ?Action5)
        (= ?Action5 AttackAction_FlashFileCompromised_5)
        (action-taken ?Action6)
        (= ?Action6 UserAction_UserOpensFlashFile_6) )
  :effect (and (action-taken ObeservedState_CVE_2010_0187_Exploited_2) ) )

(:action AttackAction_FlashFileCompromised_5
  :parameters (
      ?Action5 - Action
  )
  :precondition
      (and (action-observed ?Action5 )
           (= ?Action5 AttackAction_FlashFileCompromised_5 ) )
  :effect (and (action-taken AttackAction_FlashFileCompromised_5) ) )
```

Figure B.1: PDDL domain and problem descriptions from the CVE-2010-0187 subtree of the DoS exploit.

```
(:action UserAction_UserOpensFlashFile_6
  :parameters (
      ?Action7 - Action ;UserAction_UserUsingSocialMedia_7
  )
  :precondition
    (and
        (action-taken ?Action7)
        (= ?Action7 UserAction_UserUsingSocialMedia_7)
    ) ;and
  :effect (and (action-taken UserAction_UserOpensFlashFile_6) ) )

(:action UserAction_UserUsingSocialMedia_7
  :parameters (
      ?Action7 - Action
  )
  :precondition
      (and (action-observed ?Action7 )
           (= ?Action7 UserAction_UserUsingSocialMedia_7 ) )
  :effect (and (action-taken UserAction_UserUsingSocialMedia_7) ) )

(:action ObeservedState_CVE_2008_3111_SunJavaMultiple_Exploited_8
  :parameters (
      ?Dependency9 - software
      ?Action11 - Action ;AttackAction_JavaAppWithLongVmArgument_11
      ?Action12 - Action ;UserAction_UserStartsJavaWebStartApplication_12
  )
  :precondition
    (and
        (software-installed ?Dependency9)
        (= ?Dependency9 Sun_JRE_1_4_0_02)
        (action-taken ?Action11)
        (= ?Action11 AttackAction_JavaAppWithLongVmArgument_11)
        (action-taken ?Action12)
        (= ?Action12 UserAction_UserStartsJavaWebStartApplication_12)
    ) ;and
  :effect (and (action-taken ObeservedState_CVE_2008_3111_SunJavaMultiple_Exploited_8) ) )

(:action AttackAction_JavaAppWithLongVmArgument_11
  :parameters (
      ?Action11 - Action
  )
  :precondition
      (and (action-observed ?Action11 )
           (= ?Action11 AttackAction_JavaAppWithLongVmArgument_11 ) )
  :effect (and (action-taken AttackAction_JavaAppWithLongVmArgument_11) ) )

(:action UserAction_UserStartsJavaWebStartApplication_12
  :parameters (
      ?Action13 - Action ;UserAction_UserBrowsingInternetContent_13
  )
  :precondition
    (and
        (action-taken ?Action13)
        (= ?Action13 UserAction_UserBrowsingInternetContent_13)
    ) ;and
  :effect (and (action-taken UserAction_UserStartsJavaWebStartApplication_12) ) )
```

Figure B.1: (cont.) PDDL domain and problem descriptions from the CVE-2010-0187 subtree of the DoS exploit.

```
(:action UserAction_UserBrowsingInternetContent_13
  :parameters (
      ?Action13 - Action
  )
  :precondition
      (and (action-observed ?Action13 )
           (= ?Action13 UserAction_UserBrowsingInternetContent_13 ) )
  :effect (and (action-taken UserAction_UserBrowsingInternetContent_13) ) )

(:action ObeservedState_CVE_2010_4091_OS_Exploited_14
  :parameters (
      ?Dependency15 - software
      ?Action17 - Action ;AttackAction_PdfCompromised_17
      ?Action18 - Action ;UserAction_UserLoadsPdfDocument_18
  )
  :precondition
    (and
        (software-installed ?Dependency15)
        (= ?Dependency15 AcrobatReader_9_4_1)
        (action-taken ?Action17)
        (= ?Action17 AttackAction_PdfCompromised_17)
        (action-taken ?Action18)
        (= ?Action18 UserAction_UserLoadsPdfDocument_18)
    ) ;and
  :effect (and (action-taken ObeservedState_CVE_2010_4091_OS_Exploited_14) ) )

(:action AttackAction_PdfCompromised_17
  :parameters (
      ?Action17 - Action
  )
  :precondition
      (and (action-observed ?Action17 )
           (= ?Action17 AttackAction_PdfCompromised_17 ) )
  :effect (and (action-taken AttackAction_PdfCompromised_17) ) )

(:action UserAction_UserLoadsPdfDocument_18
  :parameters (
      ?Action13 - Action ;UserAction_UserBrowsingInternetContent_13
  )
  :precondition
    (and
        (action-taken ?Action13)
        (= ?Action13 UserAction_UserBrowsingInternetContent_13)
    ) ;and
  :effect (and (action-taken UserAction_UserLoadsPdfDocument_18) ) )

(:action UserAction_UserBrowsingInternetContent_13
  :parameters (
      ?Action13 - Action
  )
  :precondition
      (and (action-observed ?Action13 )
           (= ?Action13 UserAction_UserBrowsingInternetContent_13 ) )
  :effect (and (action-taken UserAction_UserBrowsingInternetContent_13) ) ) )
```

Figure B.1: (cont.) PDDL domain and problem descriptions from the CVE-2010-0187 subtree of the DoS exploit.

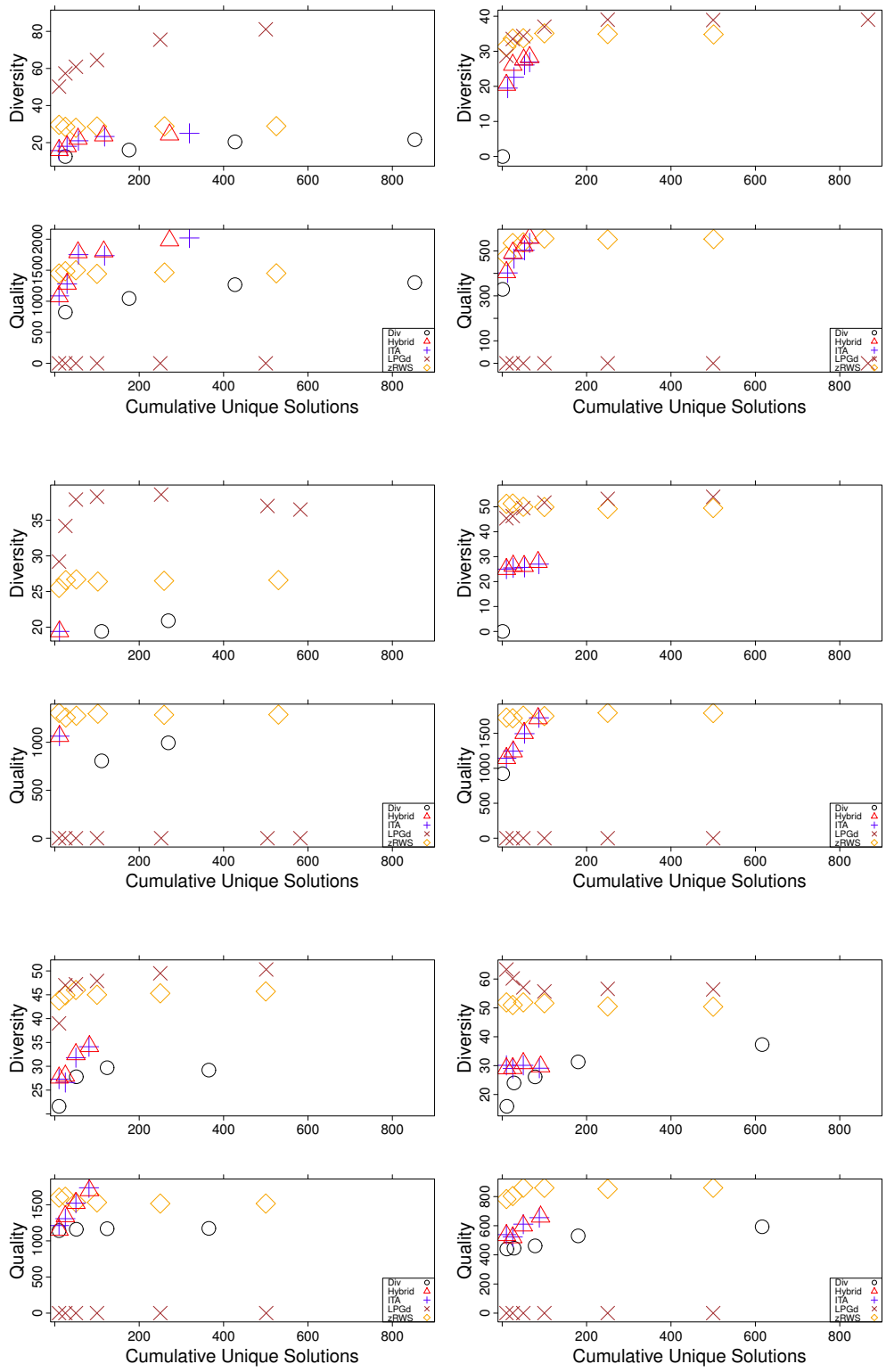

Figure B.2: Diversity (top) and Quality (bottom) over the unique plans $i \in \{10, 25, 50, 100, 250, 500, 1000\}$ for Transport problems 1 (top left), 2 (top right), 3 (center left), 4 (center right), 5 (bottom left), and 6 (bottom right).

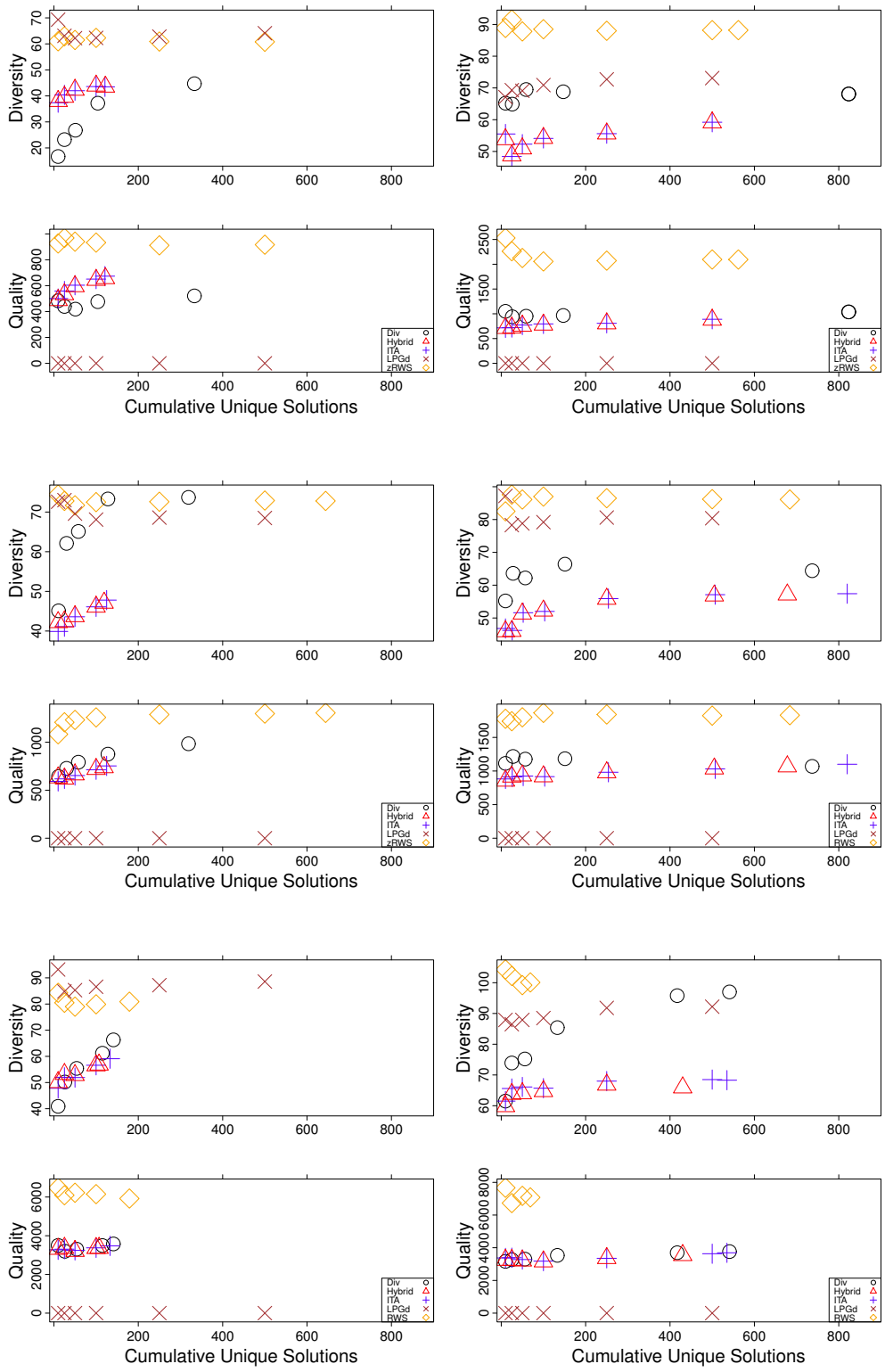

Figure B.3: Diversity (top) and Quality (bottom) over the unique plans $i \in \{10, 25, 50, 100, 250, 500, 1000\}$ for Transport problems 7 (top left), 8 (top right), 9 (center left), 10 (center right), 11 (bottom left), and 12 (bottom right).

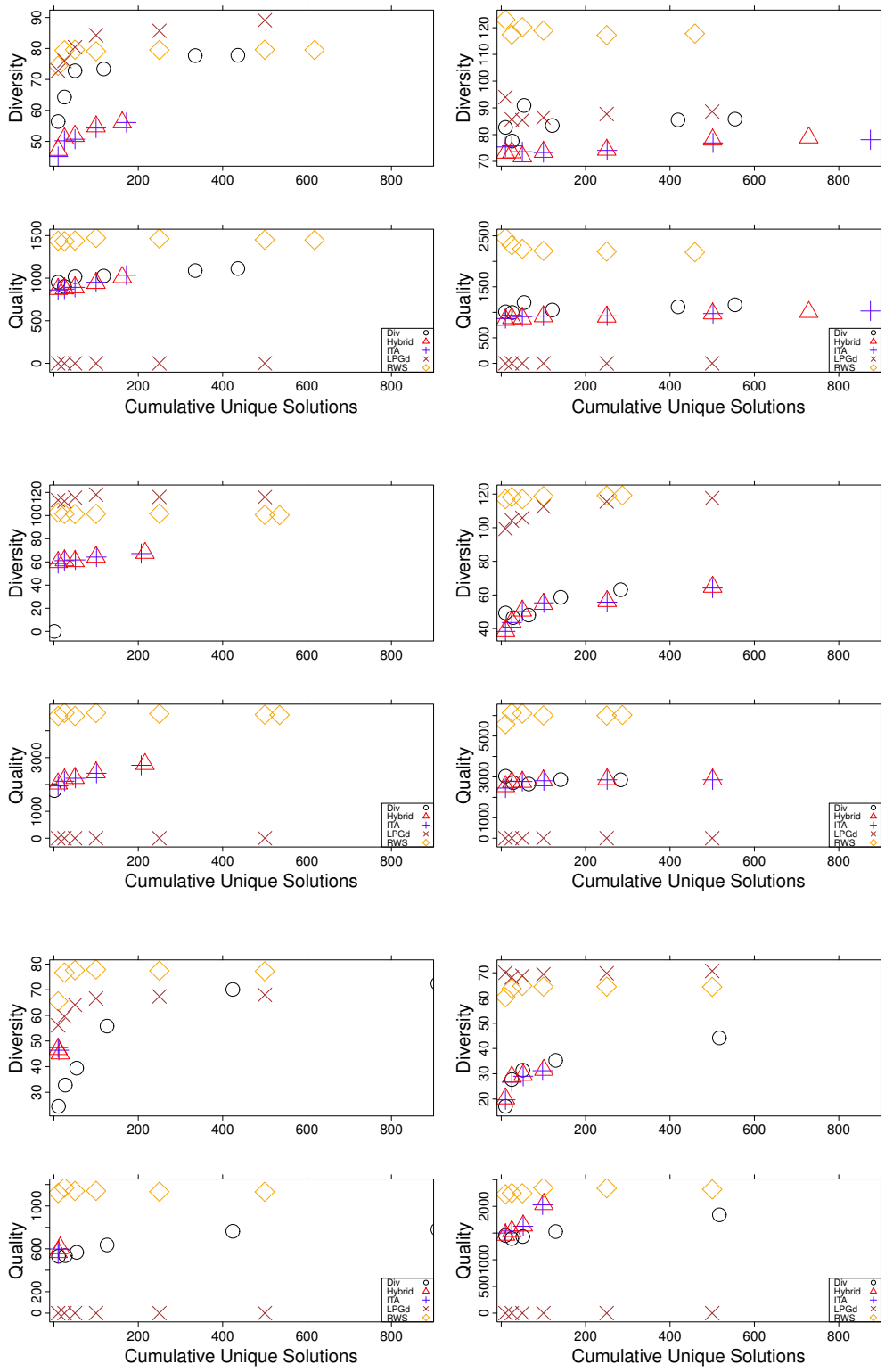

Figure B.4: Diversity (top) and Quality (bottom) over the unique plans $i \in \{10, 25, 50, 100, 250, 500, 1000\}$ for Transport problems 13 (top left), 14 (top right), 15 (center left), 16 (center right), 17 (bottom left), and 18 (bottom right).

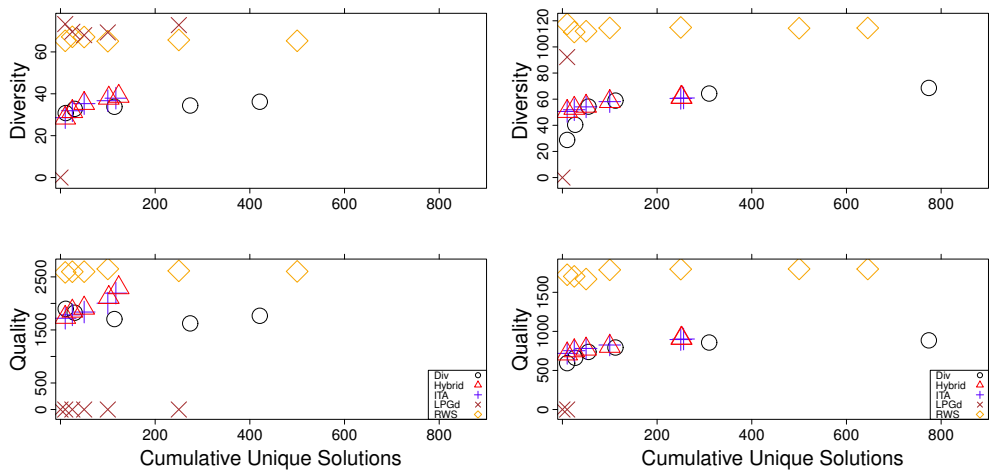

Figure B.5: Diversity (top) and Quality (bottom) over the unique plans $i \in \{10, 25, 50, 100, 250, 500, 1000\}$ for Transport problems 19 (left), 20 (right).

Appendix C
Supplemental Plots for Chapter 5

```
(define (domain synthetic-4-3-mc)
(:requirements :strips :equality
 :typing  :fluents )

(:types State )

(:predicates
  (state-active ?State - State) )

(:functions (x) (y) )

(:action Apply-0-0--1-0
  :parameters (
      ?state-0-0 - State
  )
  :precondition
    (and (state-active ?state-0-0)
         (= ?state-0-0 State-0-0))
  :effect (and
       (state-active State-1-0 )
       (increase (x) 7.0)
       (increase (y) 1012.0) ))

(:action Apply-0-0--1-1
  :parameters (
      ?state-0-0 - State
  )
  :precondition
    (and (state-active ?state-0-0)
         (= ?state-0-0 State-0-0))
  :effect (and
       (state-active State-1-1 )
       (increase (x) 510.0)
       (increase (y) 508.0) ))

(:action Apply-0-0--1-2
  :parameters (
      ?state-0-0 - State
  )
  :precondition
    (and (state-active ?state-0-0)
         (= ?state-0-0 State-0-0))
  :effect (and
       (state-active State-1-2 )
       (increase (x) 1005.0)
       (increase (y) 1.0) ))

(:action Apply-1-0--2-0
  :parameters (
      ?state-1-0 - State
  )
  :precondition
    (and (state-active ?state-1-0)
         (= ?state-1-0 State-1-0))
  :effect (and
       (state-active State-2-0 )
       (increase (x) 1.0)
       (increase (y) 1010.0) ))

(:action Apply-1-0--2-1
  :parameters (
      ?state-1-0 - State
  )
  :precondition
    (and (state-active ?state-1-0)
         (= ?state-1-0 State-1-0))
  :effect (and
       (state-active State-2-1 )
       (increase (x) 500.0)
       (increase (y) 482.0) ))

(:action Apply-1-0--2-2
  :parameters (
      ?state-1-0 - State
  )
  :precondition
    (and (state-active ?state-1-0)
         (= ?state-1-0 State-1-0))
  :effect (and
       (state-active State-2-2 )
       (increase (x) 1010.0)
       (increase (y) 1.0) ))

(:action Apply-1-1--2-0
  :parameters (
      ?state-1-1 - State
  )
  :precondition
    (and (state-active ?state-1-1)
         (= ?state-1-1 State-1-1))
  :effect (and
       (state-active State-2-0 )
       (increase (x) 1.0)
       (increase (y) 1006.0) ))

(:action Apply-1-1--2-1
  :parameters (
      ?state-1-1 - State
  )
  :precondition
    (and (state-active ?state-1-1)
         (= ?state-1-1 State-1-1))
  :effect (and
       (state-active State-2-1 )
       (increase (x) 501.0)
       (increase (y) 492.0) ))

(:action Apply-1-1--2-2
  :parameters (
      ?state-1-1 - State
  )
  :precondition
    (and (state-active ?state-1-1)
         (= ?state-1-1 State-1-1))
  :effect (and
       (state-active State-2-2 )
       (increase (x) 1001.0)
       (increase (y) 13.0) ))

(:action Apply-1-2--2-0
  :parameters (
      ?state-1-2 - State
  )
  :precondition
    (and (state-active ?state-1-2)
         (= ?state-1-2 State-1-2))
  :effect (and
       (state-active State-2-0 )
       (increase (x) 11.0)
       (increase (y) 997.0) ))

(:action Apply-1-2--2-1
  :parameters (
      ?state-1-2 - State
  )
  :precondition
    (and (state-active ?state-1-2)
         (= ?state-1-2 State-1-2))
  :effect (and
       (state-active State-2-1 )
       (increase (x) 499.0)
       (increase (y) 506.0) ))

(:action Apply-1-2--2-2
  :parameters (
      ?state-1-2 - State
  )
  :precondition
    (and (state-active ?state-1-2)
         (= ?state-1-2 State-1-2))
  :effect (and
       (state-active State-2-2 )
       (increase (x) 1002.0)
       (increase (y) 1.0) ))
```

Figure C.1: Full PDDL for the domain and problem $4{\times}3^{x}_{nil}$.

```
(:action Apply-2-0--3-0
  :parameters (
      ?state-2-0 - State
  )
  :precondition
      (and (state-active ?state-2-0)
           (= ?state-2-0 State-2-0))
  :effect (and
          (state-active State-3-0 )
          (increase (x) 7.0)
          (increase (y) 995.0) ))

(:action Apply-2-0--3-1
  :parameters (
      ?state-2-0 - State
  )
  :precondition
      (and (state-active ?state-2-0)
           (= ?state-2-0 State-2-0))
  :effect (and
          (state-active State-3-1 )
          (increase (x) 506.0)
          (increase (y) 502.0) ))

(:action Apply-2-0--3-2
  :parameters (
      ?state-2-0 - State
  )
  :precondition
      (and (state-active ?state-2-0)
           (= ?state-2-0 State-2-0))
  :effect (and
          (state-active State-3-2 )
          (increase (x) 986.0)
          (increase (y) 1.0) ))

(:action Apply-2-1--3-0
  :parameters (
      ?state-2-1 - State
  )
  :precondition
      (and (state-active ?state-2-1)
           (= ?state-2-1 State-2-1))
  :effect (and
          (state-active State-3-0 )
          (increase (x) 1.0)
          (increase (y) 1017.0) ))

(:action Apply-2-1--3-1
  :parameters (
      ?state-2-1 - State
  )
  :precondition
      (and (state-active ?state-2-1)
           (= ?state-2-1 State-2-1))
  :effect (and
          (state-active State-3-1 )
          (increase (x) 488.0)
          (increase (y) 505.0) ))

(:action Apply-2-1--3-2
  :parameters (
      ?state-2-1 - State
  )
  :precondition
      (and (state-active ?state-2-1)
           (= ?state-2-1 State-2-1))
  :effect (and
          (state-active State-3-2 )
          (increase (x) 994.0)
          (increase (y) 4.0) ))

(:action Apply-2-2--3-0
  :parameters (
      ?state-2-2 - State
  )
  :precondition
      (and (state-active ?state-2-2)
           (= ?state-2-2 State-2-2))
  :effect (and
          (state-active State-3-0 )
          (increase (x) 5.0)
          (increase (y) 1008.0) ))

(:action Apply-2-2--3-1
  :parameters (
      ?state-2-2 - State
  )
  :precondition
      (and (state-active ?state-2-2)
           (= ?state-2-2 State-2-2))
  :effect (and
          (state-active State-3-1 )
          (increase (x) 489.0)
          (increase (y) 504.0) ))

(:action Apply-2-2--3-2
  :parameters (
      ?state-2-2 - State
  )
  :precondition
      (and (state-active ?state-2-2)
           (= ?state-2-2 State-2-2))
  :effect (and
          (state-active State-3-2 )
          (increase (x) 1015.0)
          (increase (y) 1.0) ))

(:action Apply-3-0--4-0
  :parameters (
      ?state-3-0 - State
  )
  :precondition
      (and (state-active ?state-3-0)
           (= ?state-3-0 State-3-0))
  :effect (and
          (state-active State-4-0 )
          (increase (x) 1.0)
          (increase (y) 988.0) ))

(:action Apply-3-1--4-0
  :parameters (
      ?state-3-1 - State
  )
  :precondition
      (and (state-active ?state-3-1)
           (= ?state-3-1 State-3-1))
  :effect (and
          (state-active State-4-0 )
          (increase (x) 505.0)
          (increase (y) 503.0) ))

(:action Apply-3-2--4-0
  :parameters (
      ?state-3-2 - State
  )
  :precondition
      (and (state-active ?state-3-2)
           (= ?state-3-2 State-3-2))
  :effect (and
          (state-active State-4-0 )
          (increase (x) 1003.0)
          (increase (y) 5.0) ))
) ;end of define
```

Figure C.1: (continued) Full PDDL for the domain and problem $4{\times}3^{x}_{nil}$.

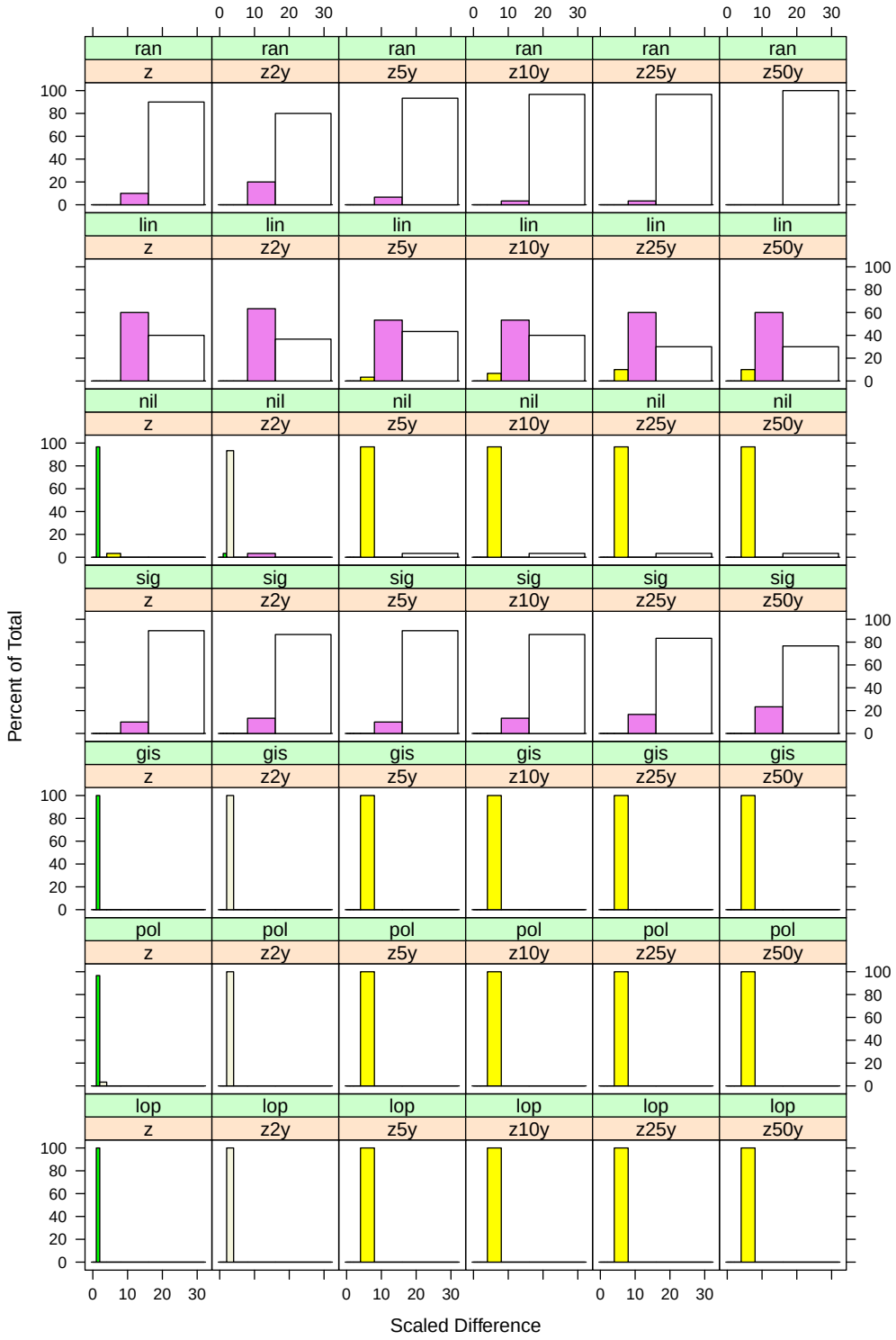

Figure C.2: Log-Histograms of the scaled difference from the minimum for the z_{*y} problems. Bins for the bottom axis are set at $\{0, 1, 2, 4, 8, 16, \geq 32\}$ to provide a visual representation of how well an algorithm does. Better performance is indicated by thinner and taller bars to the left. For more details, see the discussion of Figure 5.2.

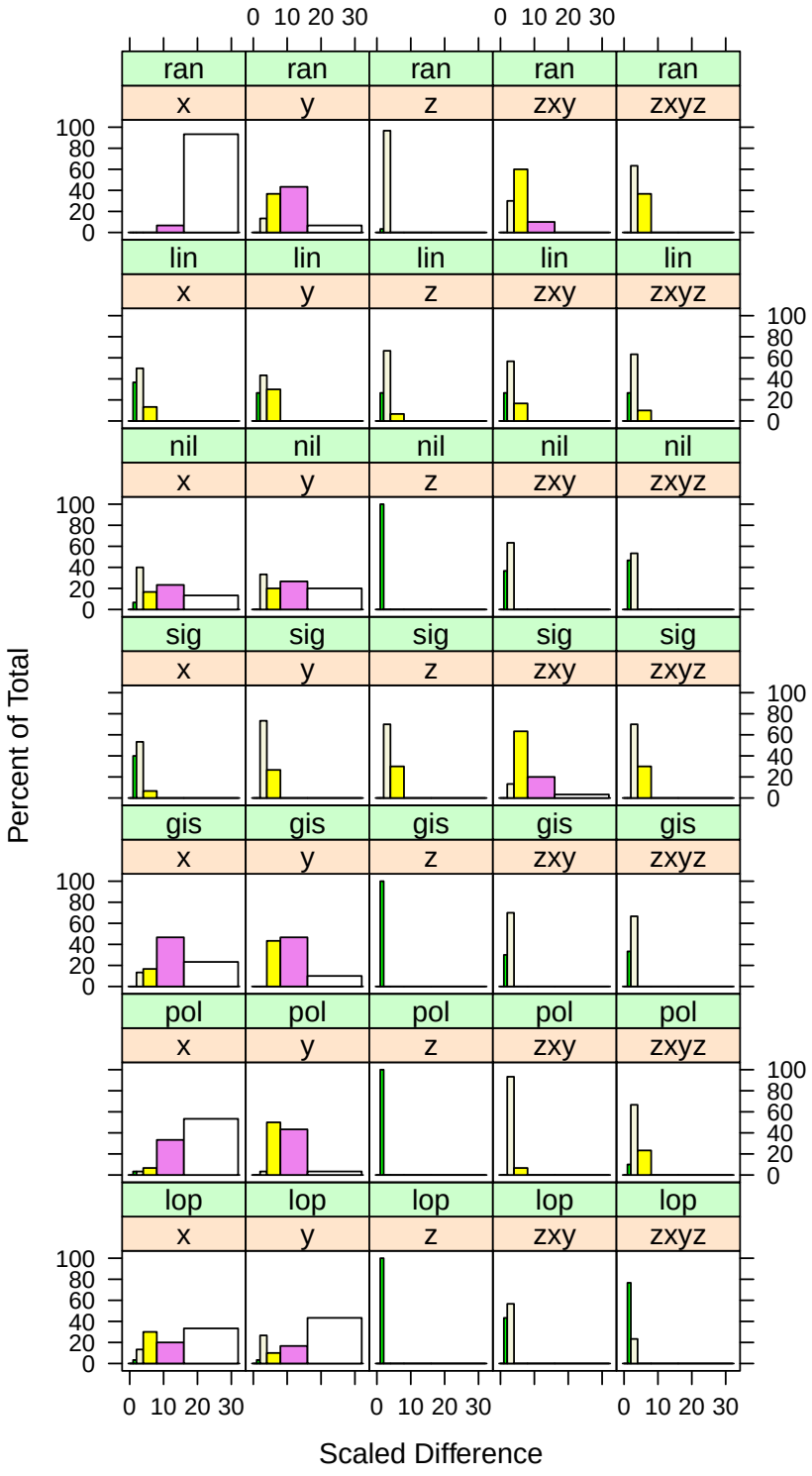

Figure C.3: Log-Histograms of the scaled difference from the minimum for *MQA* to generate a single solution in 30 problems of `BiSyn`.

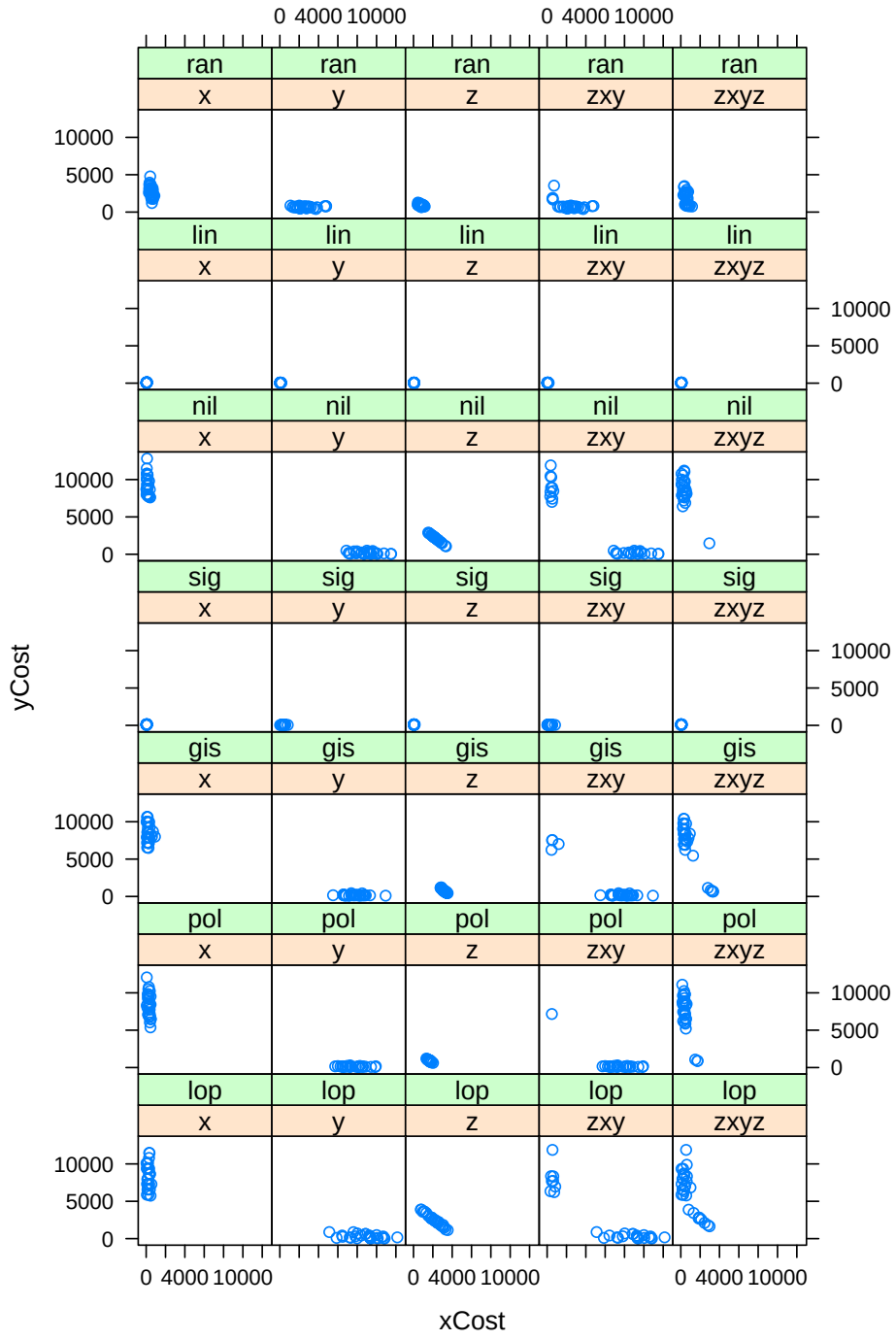

Figure C.4: Solution xy-interaction plots for *MQA* to generate a single solution in 30 problems of `BiSyn`.

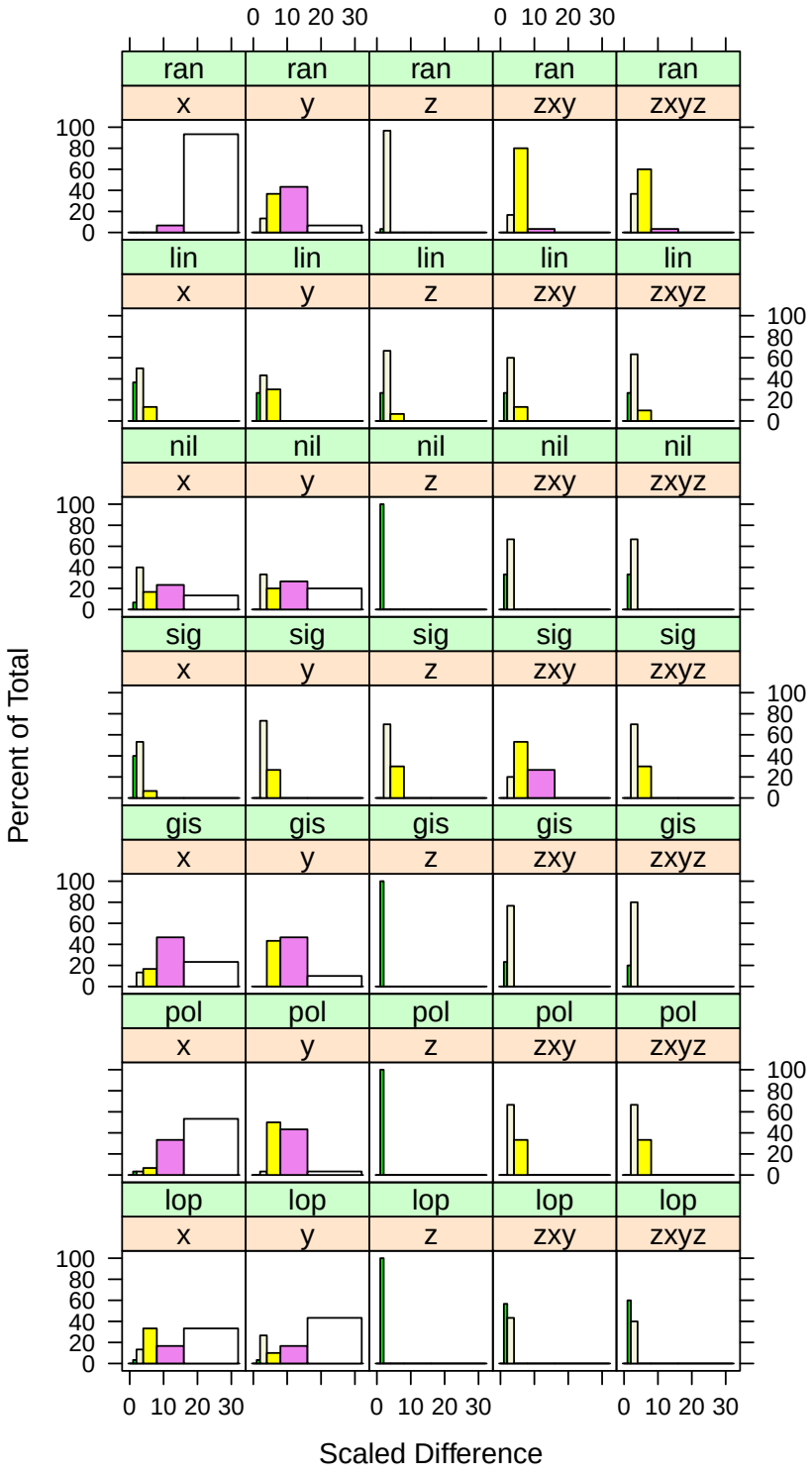

Figure C.5: Log-Histograms of the scaled difference from the minimum for MQA_D to generate a single solution in 30 problems of `BiSyn`.

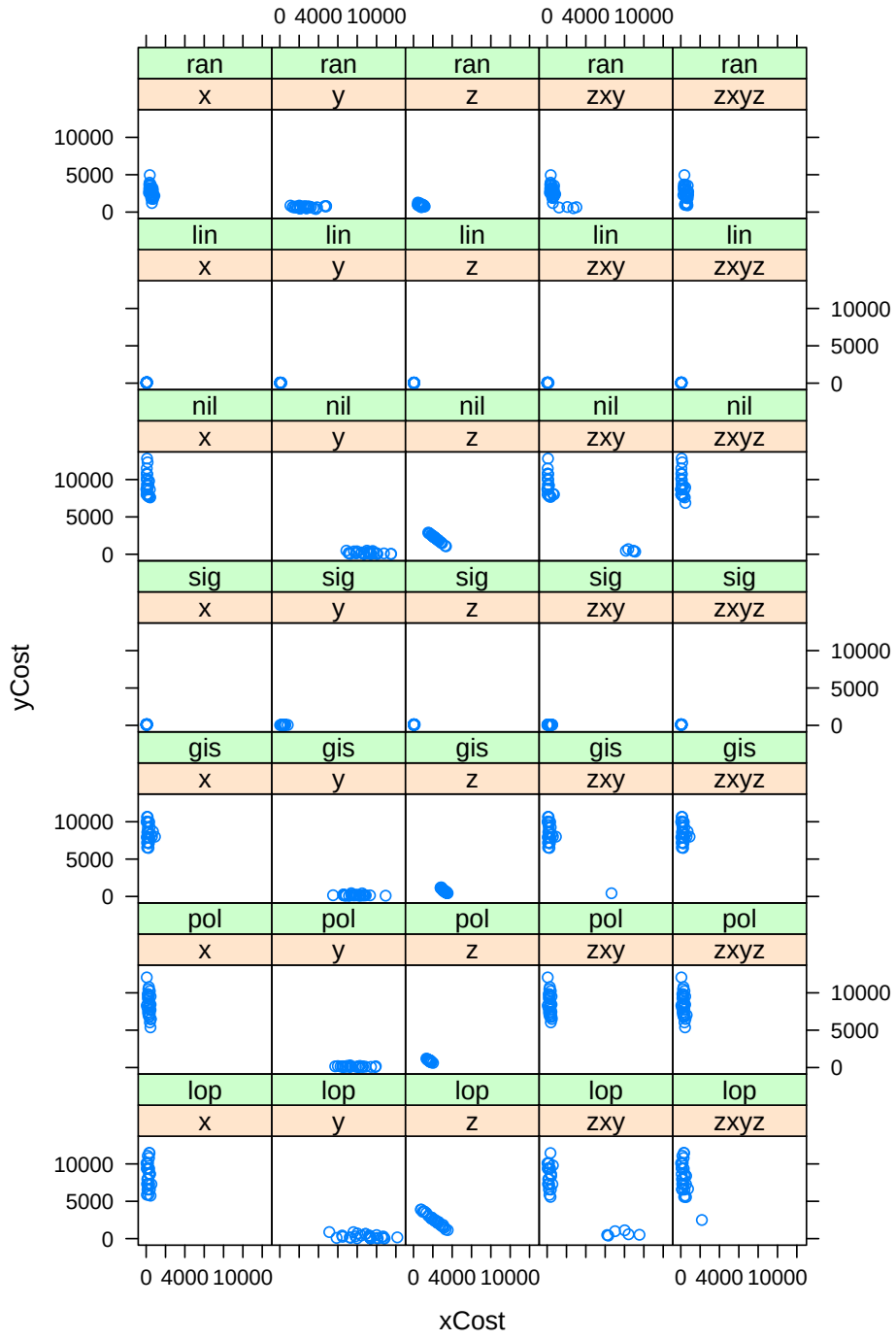

Figure C.6: Solution xy-interaction plots for MQA_D to generate a single solution in 30 problems of `BiSyn`.

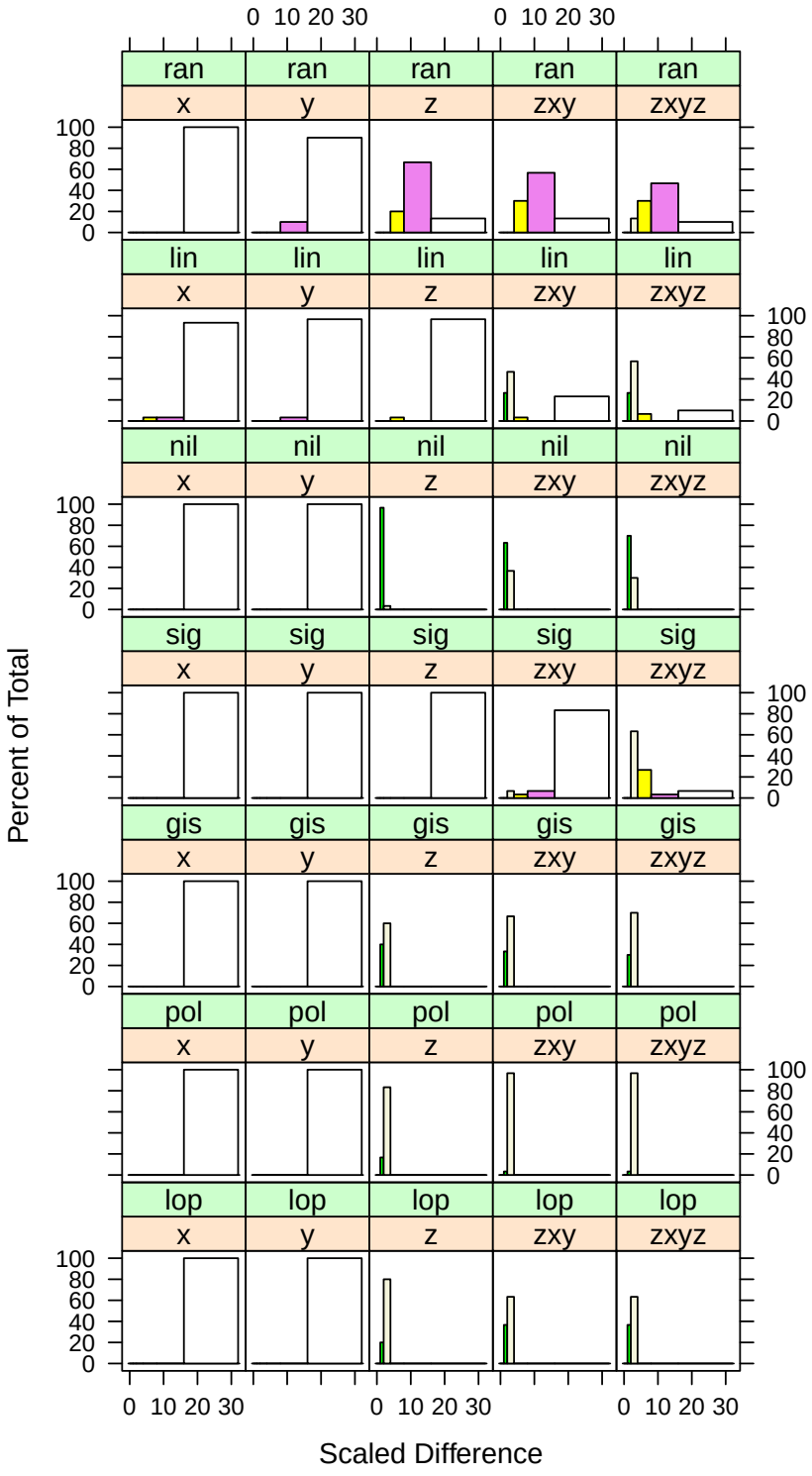

Figure C.7: Log-Histograms of the scaled difference from the minimum for MQA_S to generate a single solution in 30 problems of `BiSyn`.

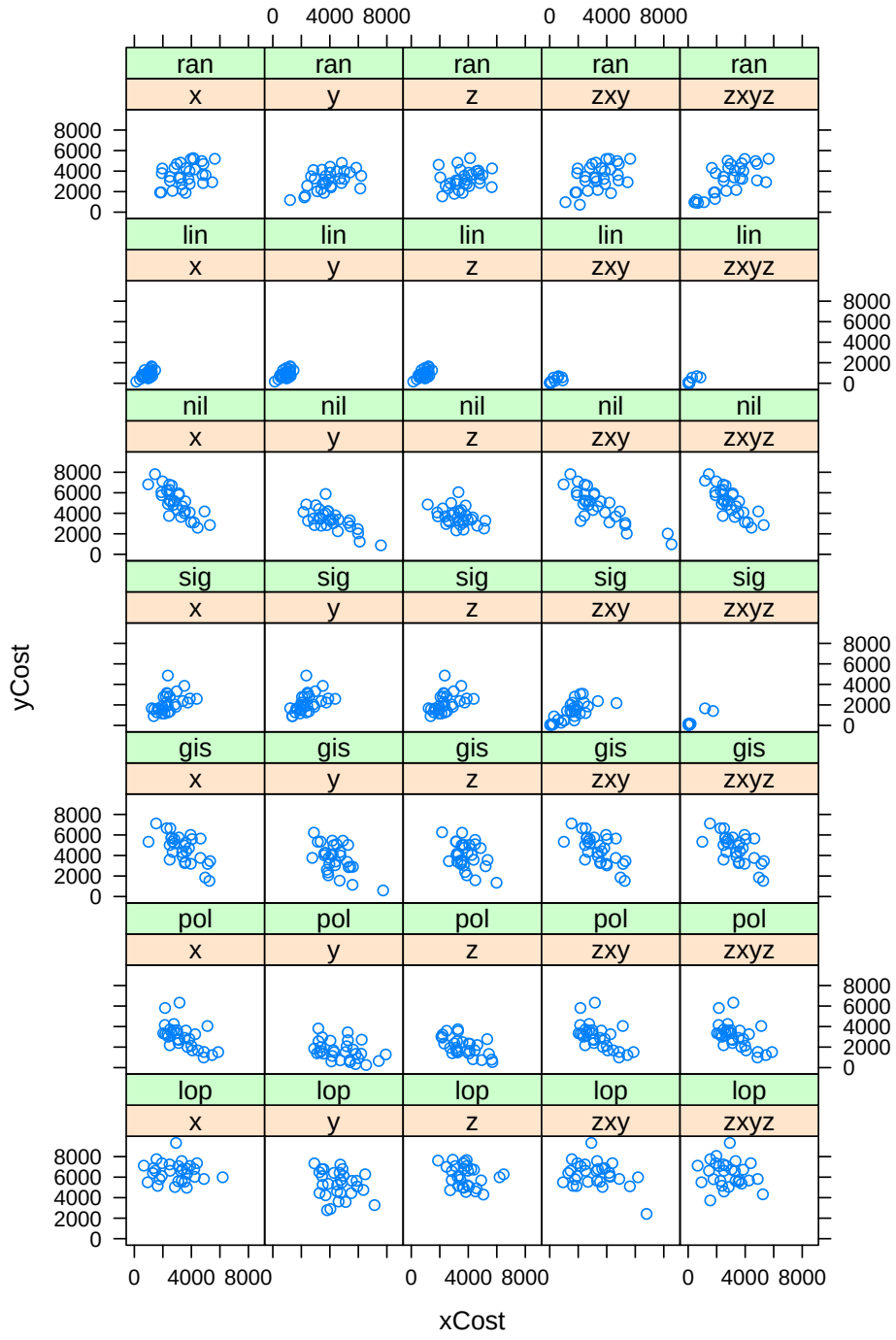

Figure C.8: Solution xy-interaction plots for MQA_S to generate a single solution in 30 problems of `BiSyn`.

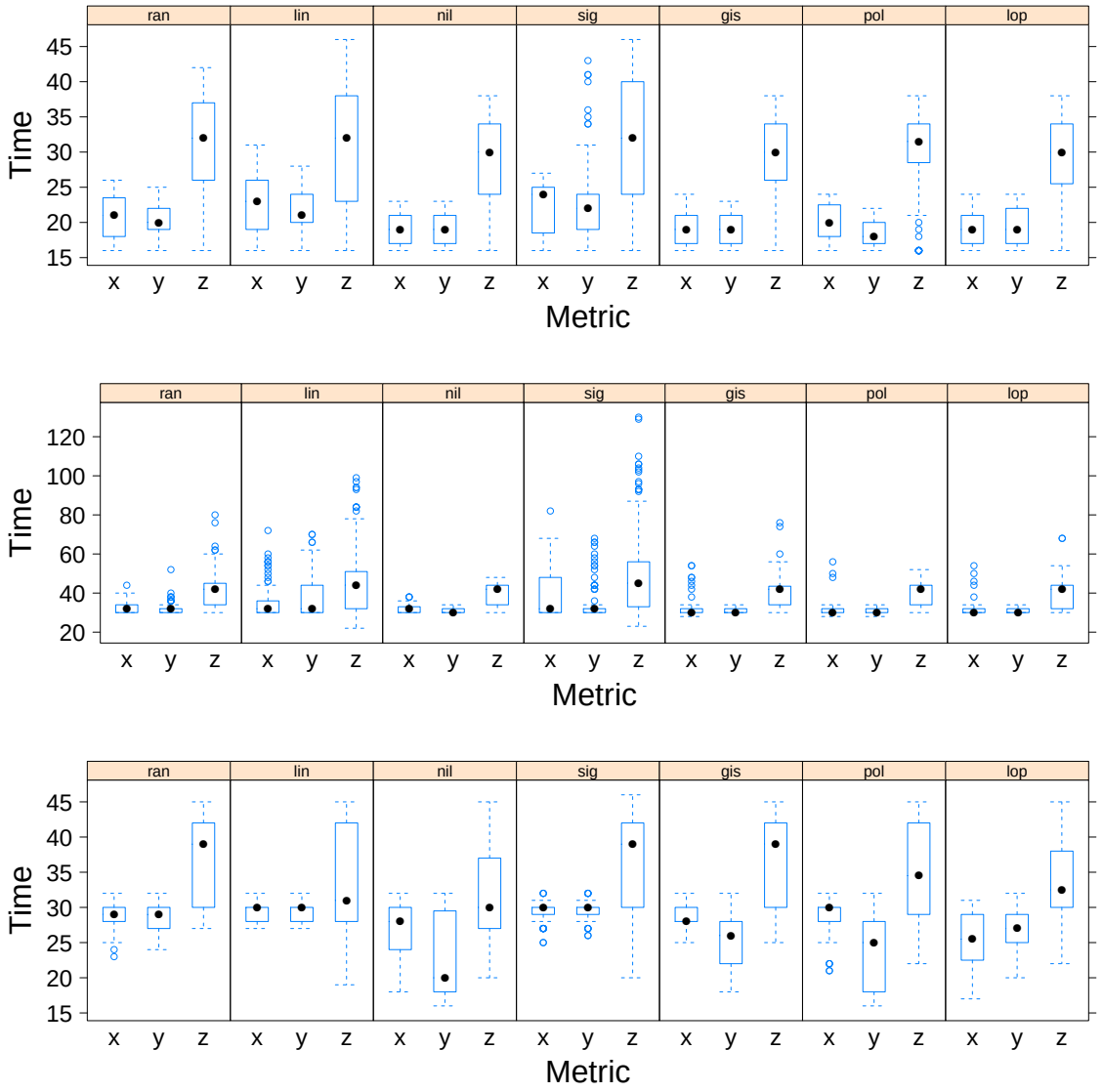

Figure C.9: Runtime distributions of all functions and metrics for MQA_T (top) and MQA_{TD} (bottom) to generate a 100 solutions in p00 of `BiSyn`.

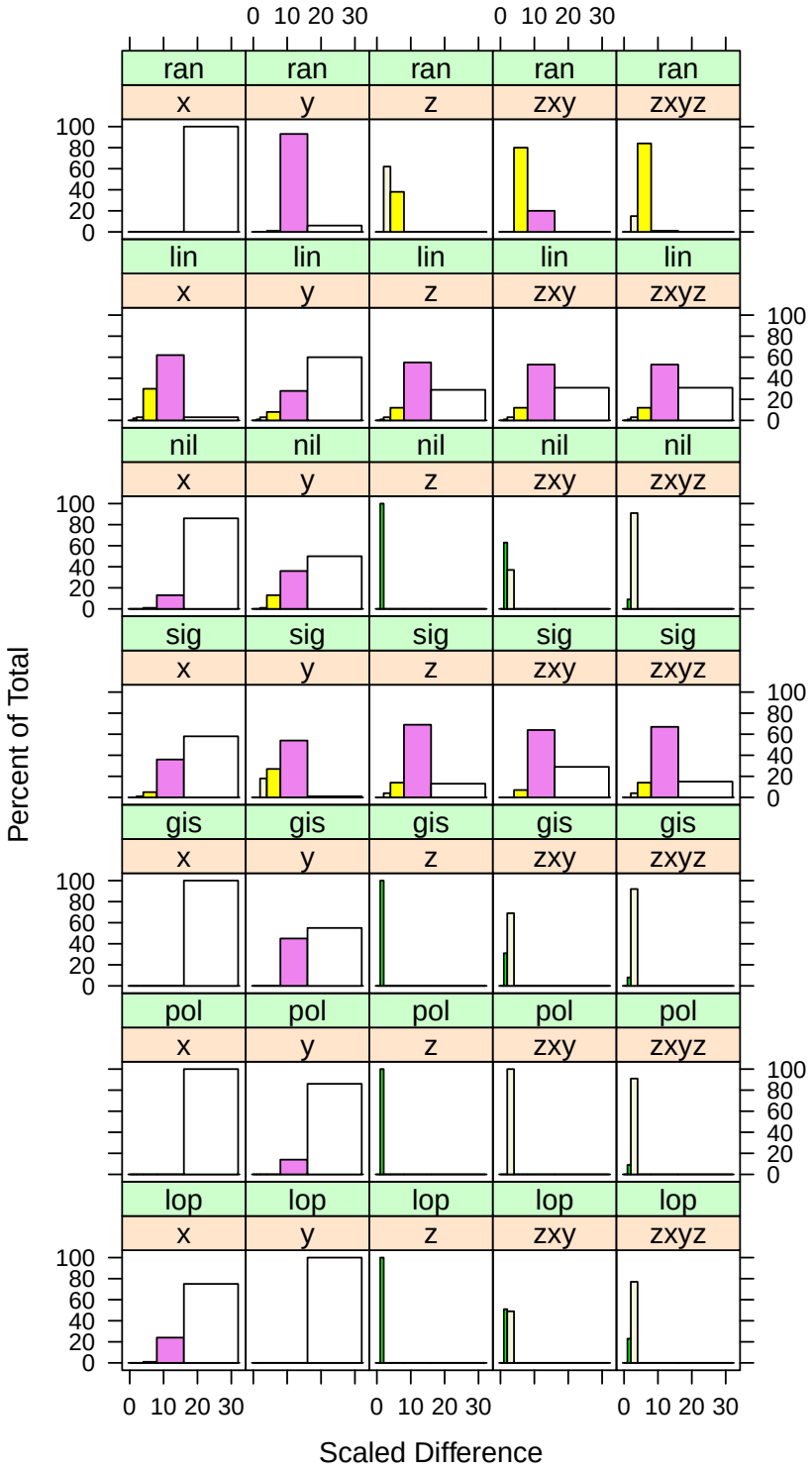

Figure C.10: Log-Histograms of the scaled difference from the minimum for MQA_T to generate a 100 solutions in p00 problems of `BiSyn`.

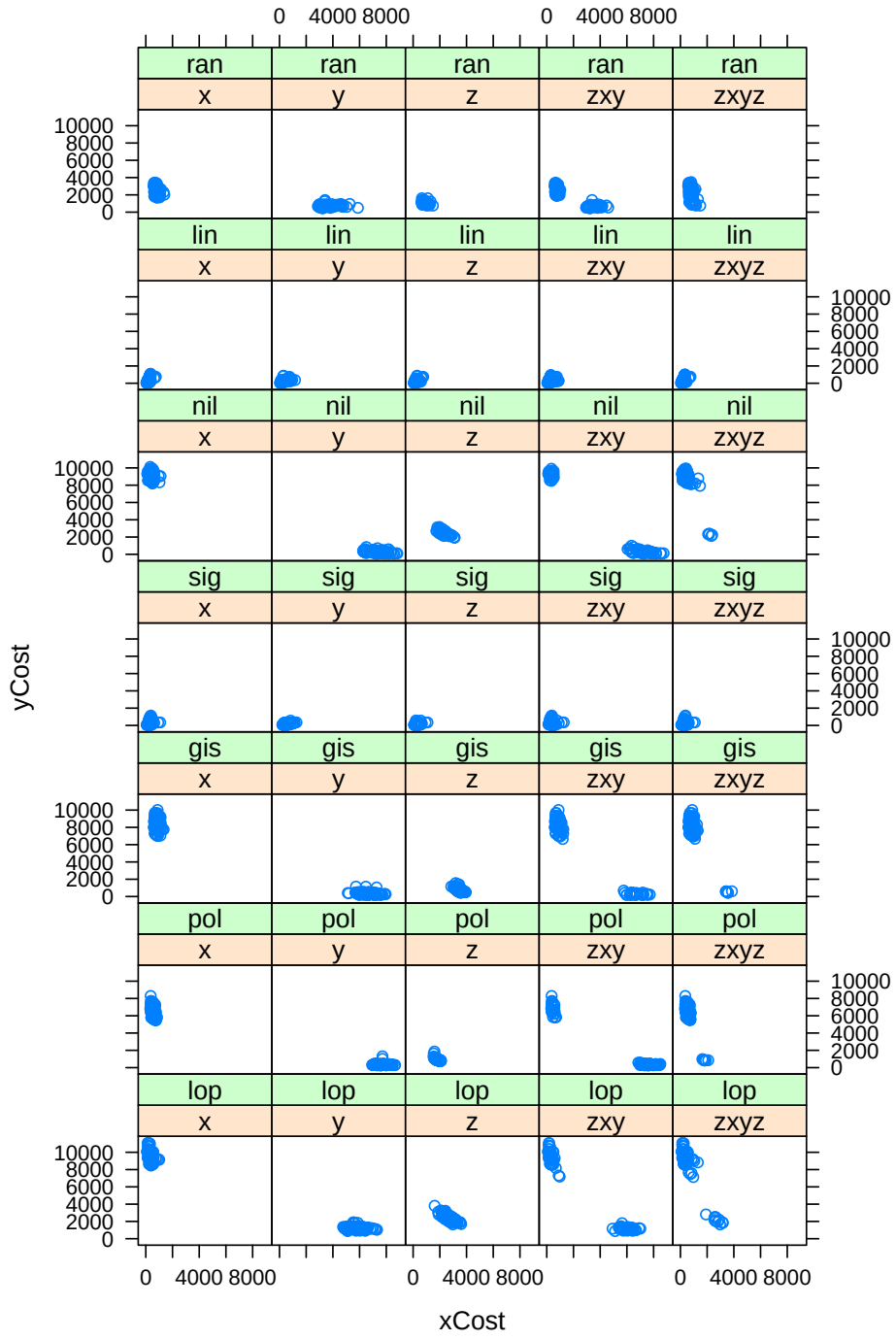

Figure C.11: Solution xy-interaction plots for MQA_T to generate a 100 solutions in p00 problems of `BiSyn`.

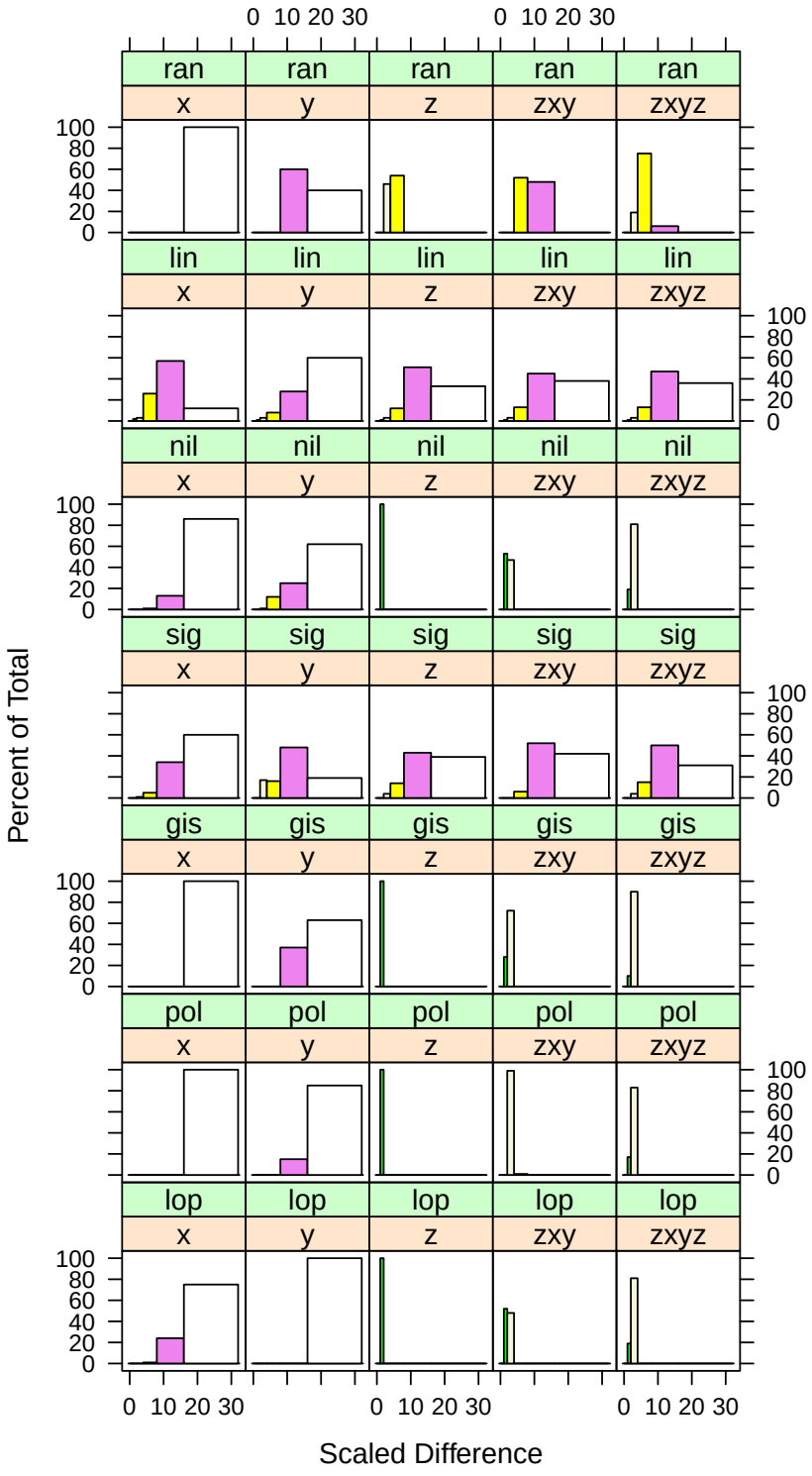

Figure C.12: Log-Histograms of the scaled difference from the minimum for MQA_{TD} to generate a 100 solutions in p00 problems of `BiSyn`.

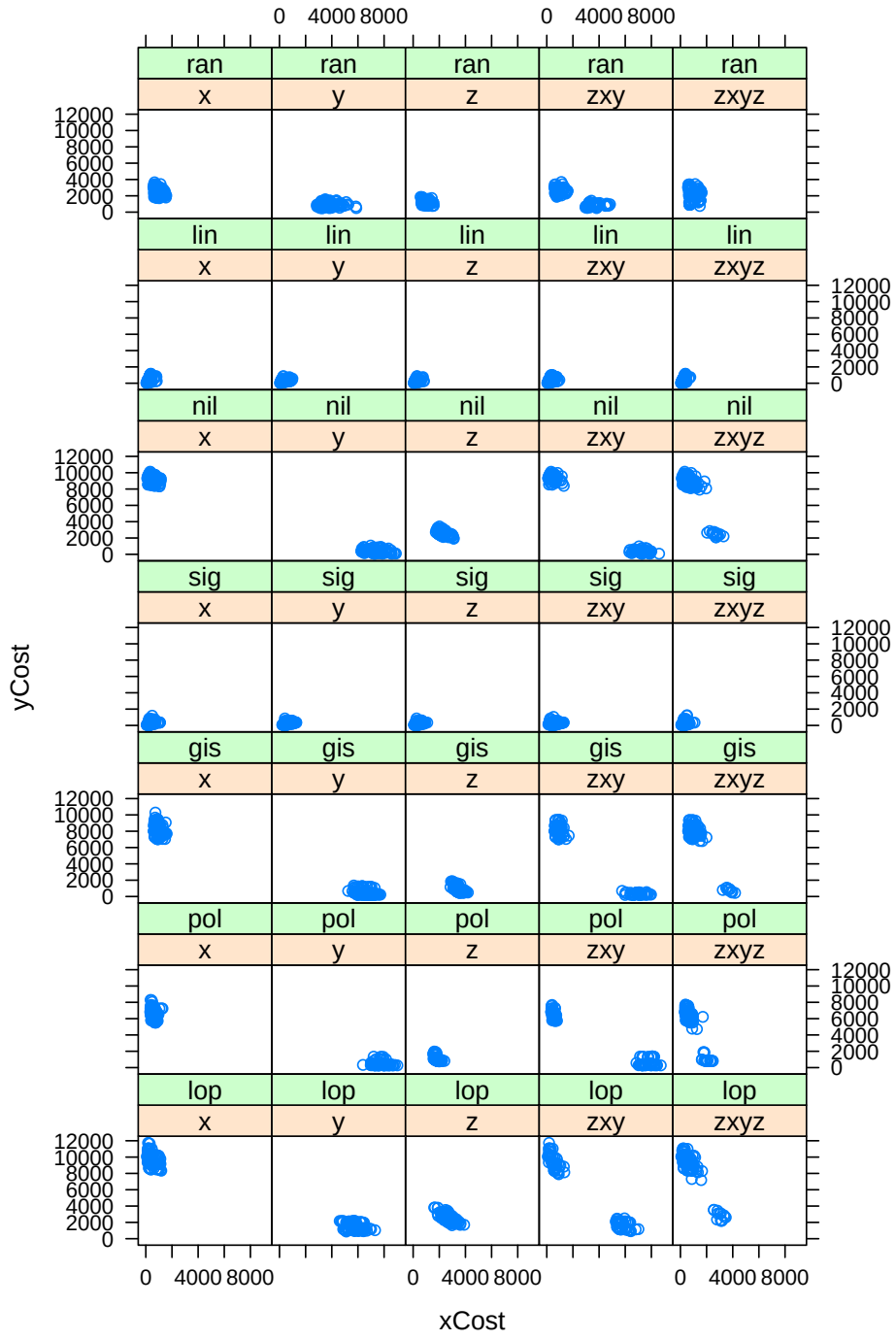

Figure C.13: Solution xy-interaction plots for MQA_{TD} to generate a 100 solutions in p00 problems of `BiSyn`.

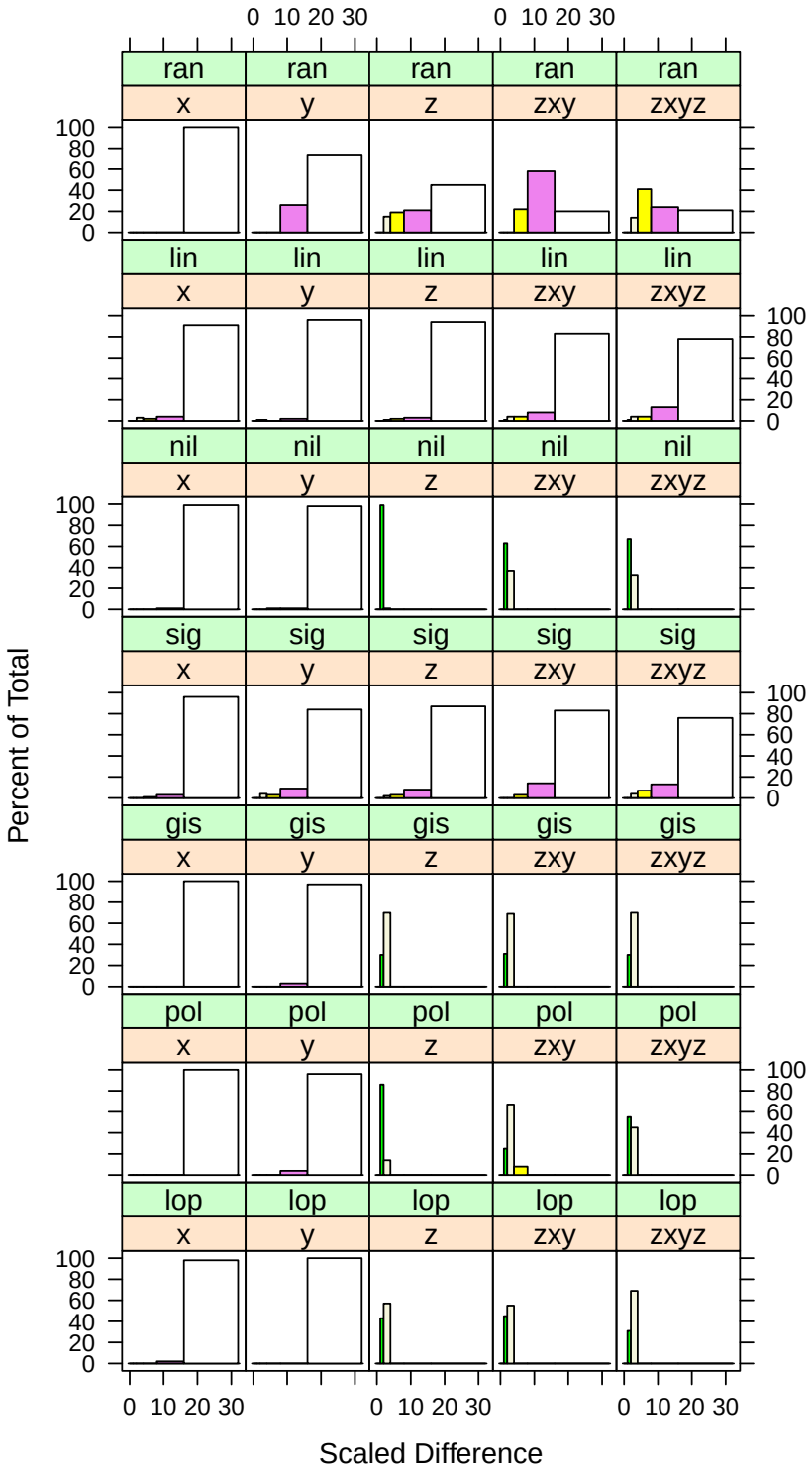

Figure C.14: Log-Histograms of the scaled difference from the minimum for MQA_{TS} to generate a 100 solutions in p00 problems of `BiSyn`.

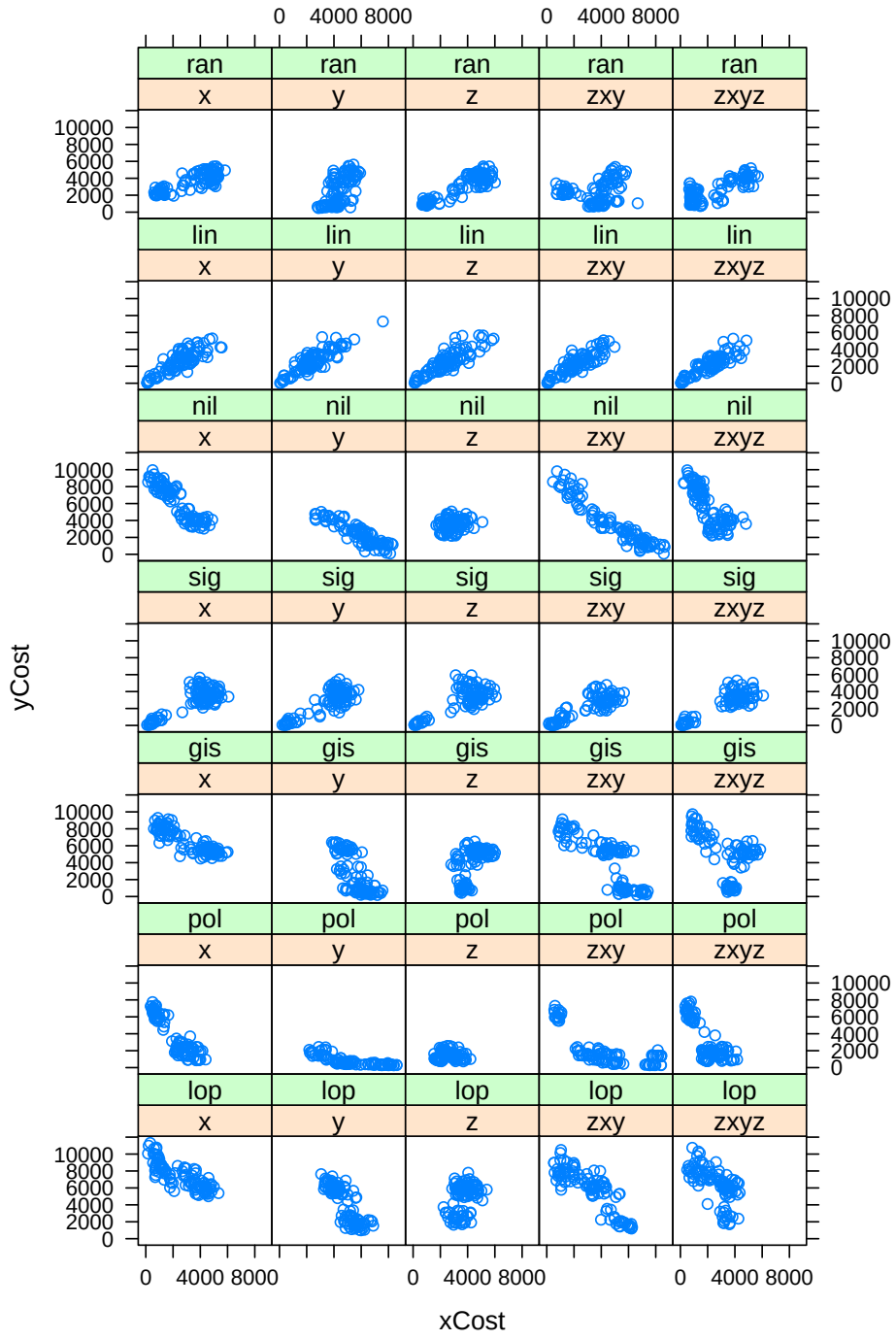

Figure C.15: Solution xy-interaction plots for MQA_{TS} to generate a 100 solutions in p00 problems of `BiSyn`.